Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Gestión de QoS para servicios multimedia

TITULACIÓN: Ingeniería Técnica de Telecomunicación, especialidad Telemática

AUTOR: Pau Muñoz Montesinos

DIRECTOR: Antoni Oller Arcas

DATA: 2 de julio de 2009

Título: Gestión de QoS para servicios multimedia

Autor: Pau Muñoz Montesinos

Director: Antoni Oller Arcas

Data: 2 de julio de 2009

Resumen

Las redes de comunicaciones y las máquinas que las forman están evolucionando hacia sistemas de mayora capacidad y potencia, creando un nicho para el desarrollo de nuevas tecnologías y servicios.

Uno de los grupos de servicios más solicitados son los que tienen alguna relación con los contenidos multimedia. Estos servicios se centran en el acceso a los contenidos, pero no tienen en cuenta ningún parámetro externo para asegurar su llegada ni para la optimización de los recursos de la red.

Para ello es necesario incluir utilidades que tomen en cuenta el contexto en que se va a prestar el servicio para intentar asegurar el envío de los datos de forma correcta, optima y procurando no afectar al resto de comunicaciones. En otras palabras, proporcionar calidad de servicio.

El objetivo de este proyecto es investigar esta problemática y diseñar una infraestructura que permita la gestión de la calidad de servicio ofrecida a cada usuario.

Inicialmente, se ha seleccionado un servio multimedia que pudiera aprovecharse de la infraestructura que se ha diseñado. Posteriormente, se han investigado tecnologías que fueran capaces de obtener información para definir a cada usuario y la calidad de servicio que se le debe proporcionar.

El paso siguiente ha consistido en implementar y realizar pruebas para definir qué tecnologías se adaptan mejor a las necesidades del servicio, al escenario de trabajo y cuales proporcionan mejores resultados. Se han diseñado los módulos básicos del sistema y se han establecido las funcionalidades que proporcionaría cada uno de esos módulos, aunque estas han ido variando debido a las necesidades que iban apareciendo a medida que se implementaba cada una de ellas.

Finalmente, se ha implementado una utilidad capaz de determinar la calidad de servicio a ofrecer a cada usuario y se ha desplegado un elemento que permite prestar el servicio seleccionado en función de las decisiones de esta utilidad.

Title QOS management for multimedia services

Author: Pau Muñoz Montesinos

Director: Antoni Oller Arcas

Date: June, 30th 2009

Overview

Communication networks and machines that are part of them are evolving to more powerful systems and with more network capacity, building a new space to develop new technologies and services.

One of the most required services are those which are related with media contents. These services are focused in granting access to those contents, but don't care about external parameters to guarantee its reception nor network resource optimization.

To achieve this objective, it's need to include utilities that care about the context where the services will be provided trying to guarantee the correct and optimal data transfer and attempting not to disturb other communications. In other words, providing quality of service.

The objectives of this project are researching about these problems and design a system that can manage quality if service provided to each user.

Beginning, a multimedia service which can take advantage of the designed infrastructure has been chosen. After that, technologies able to obtain information to define each user and the quality of service it should be provided with.

The next step consisted in implementing and testing those technologies to select those which fit better to the needs of the service, the developing scenario and which provide better results. Basic system modules and functionalities have been designed and established, although they've changed due to new requirements appeared during their implementation.

Finally, a utility to decide the quality of service provided to each user has been implemented and an element able to provide the selected service depending on this utility decision has been deployed.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Las redes de comunicación están sufriendo una evolución, pasando a tener cada día mayores tasas de transferencia y una creciente calidad de transmisión. Así mismo, las máquinas que las forman también están sufriendo mejoras, convirtiéndose en dispositivos con una mayor potencia de procesado, capacidad de almacenamiento, etc. Estos cambios están creando un interés creciente en la industria por desarrollar nuevas aplicaciones, tecnologías y servicios que sean capaces de sacarles partido.

Entre los servicios más solicitados por los usuarios (y que por tanto, pueden tener una mayor demanda a la hora de ser comercializados) se encuentran los relacionados con la industria multimedia. Normalmente se trata de servicios que facilitan al usuario el acceso a contenidos multimedia remotos, su visualización en diversos dispositivos o que dan acceso a contenidos en el instante en que el usuario desee.

Prácticamente ninguno de estos servicios asegura un funcionamiento en el que se minimicen los errores o las interrupciones. Tan solo sirven aquello que se les pide. Para poder cumplir esas necesidades requieren de utilidades transparentes [7] que adapten la calidad de la comunicación, de sistemas que aprovechen de forma más eficiente los recursos disponibles o de una combinación de ambos.

La mayoría de estas utilidades pretenden proporcionar lo que en el mundo de las telecomunicaciones se conoce como calidad de servicio (QOS). Se ofrecen servicios más especializados gracias a un mayor conocimiento del usuario que utiliza el servicio multimedia y el entorno en el que se encuentra.

Para poder prestar estos servicios es necesario obtener esta información sobre el usuario [9]. También es necesario poder almacenar la información y saber cómo procesarla. Y finalmente se requiere de un servicio base que pueda ser mejorado con estas nuevas características.

El objetivo de este proyecto es ahondar en esta problemática, para poder presentar soluciones viables a cada una de las necesidades presentadas.

Este documento es una recopilación del trabajo que se ha realizado para cumplir con estos objetivos, y su distribución es la siguiente.

En el primer capítulo se describe un servicio multimedia que carece de calidad de servicio y que podría beneficiarse de las soluciones que se buscan con este proyecto. Se focaliza la problemática en único escenario para poder dar un resultado concreto, ya que si se intentara abarcar todos los escenarios posibles el proyecto se volvería tremendamente complejo, demasiado difuso o simplemente no tendría fin. También se presentan las primeras pinceladas de los que podrían ser algunos caminos hacia los objetivos marcados.

En el segundo capítulo se entra ya en una descripción más exhaustiva de los elementos necesarios para implementar cada una de las soluciones previamente propuestas, así como del proceso seguido para su desarrollo. También se presenta el espacio en el que se ha trabajado y las herramientas utilizadas.

El tercer capítulo presenta la planificación de tareas que se ha hecho y una evaluación de los costes de desarrollo del proyecto.

Para terminar, el capítulo final recoge las conclusiones que se han extraído mientras se realizaba el trabajo, una propuesta de evolución para el diseño presentado y una evaluación del impacto medioambiental que tiene el proyecto.

Capítulo 1. DESCRIPCIÓN DEL PROYECTO

Existe una gran variedad de servicios multimedia que ofrecer a los usuarios. Voz sobre IP, videoconferencia, media en *streaming*, son solo algunos de ellos.

Este proyecto centra su atención en uno de los servicios más comunes en estos tiempos: el video bajo demanda. En el video bajo demanda, el usuario puede elegir qué programa desea ver y posteriormente iniciar, detener e incluso continuar su visualización cuando él lo desee. Estos servicios se suelen implementar en los extendidos sistemas de televisión en pago por visión.

1.1. Video bajo demanda: definición y problemática

En los sistemas de video bajo demanda, se ofrece a los clientes la posibilidad de seleccionar un contenido de un repositorio de contenidos disponibles. Una vez que selecciona el contenido, éste debe ser enviado al cliente.

Esto significa que debe haber como mínimo tres elementos: un repositorio de contenidos en el que se almacenará todo el material que se quiera ofrecer; un servidor de contenidos capaz de enviar los videos al cliente; y un servidor dotado de la inteligencia necesaria para poderse comunicar con los clientes y localizar los contenidos.

El cliente tan solo necesita disponer de un dispositivo que se pueda comunicar con el servidor para pedir los contenidos y que sea capaz de mostrar los videos enviados por el servidor de contenidos.

Los problemas de este sistema provienen de la gran variedad de dispositivos desde los que un cliente puede pedir un contenido, desde ordenadores a teléfonos móviles, pasando por decodificadores de televisión por cable y otros muchos; así como de la multiplicidad de entornos en los que éste se puede encontrar, como pueden ser redes compartidas, conexiones por satélite, comunicaciones por redes de telefonía móvil, etc.

Con una variedad tan grande de posibles destinos, parece evidente que enviar del mismo modo el mismo contenido a todos ellos no puede resultar en un mismo resultado para todos. En algunos casos, se podría estar utilizando una gran cantidad de ancho de banda para enviar un video a un simple dispositivo móvil, incapaz de mostrar toda esa información. En otros, se podría estar enviando un exceso de datos por un enlace demasiado pequeño, de manera que el resultado quedaría muy dañado. Y en cualquier caso, se estaría introduciendo una cantidad ingente de tráfico innecesario en la red que iría degradando la calidad del resto de servicios que se encontraran en la red.

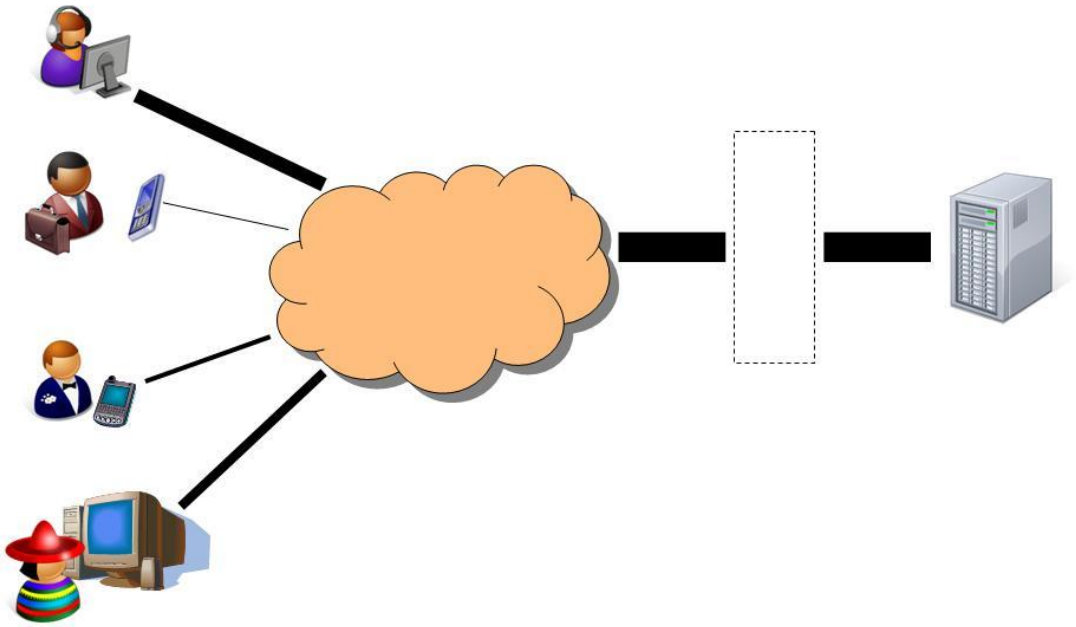

Fig. 1.1 Diferentes terminales y entornos que se conectan a un repositorio a través de la red. El espacio en blanco representa una posible situación para un elemento de gestión de QOS.

En el escenario (Fig. 1.1) se ven representados los elementos que se han descrito. Se nombran de nuevo a continuación listando sus características:

- Los usuarios:
  - Heterogeneidad de dispositivos: La característica principal de los usuarios es que, al tratarse de un público muy amplio y poco especializado, cada uno de ellos dispone de un dispositivo que se adapta a él, y no necesariamente a la necesidad que se está intentando cubrir. La gran variedad de dispositivos se diferencia en aspectos tan variados como pueden ser los tamaños de pantalla, el procesador, el software, etc.
- Variación de las características del medio y de las múltiples de tecnologías de acceso:
  - Acceso a medios compartidos, no compartidos, a medios dirigidos (cable) o no dirigidos (como el aire), etc.
  - Colisiones, tasa de errores, inmunidad al ruido, retardo introducido, etc.
- Red entre el dispositivo del usuario y el servidor de contenidos:
  - Número y características de dispositivos intermedios, existencia de elementos de traducción de direcciones, etc.
- Servidor de contenidos:
  - Servicio de contenidos en descarga: El contenido no se visualizará hasta haber recibido la totalidad del mismo.

- Servicio de contenidos en *streaming*: El contenido se puede empezar a reproducir a medida que se recibe.

Por tanto, el problema está en cómo definir al usuario. Dado que ya se han dedicado varios proyectos dedicados a determinar las características teóricas de los diferentes dispositivos existentes, la aproximación que se ha decidido seguir en este proyecto es la de determinar las características de la red del usuario.

Existen diversos tipos de técnicas para determinar los parámetros de la red del cliente y las limitaciones en su camino hasta el servidor. Algunos utilizan el tráfico ya existente entre estos dos puntos para obtener la información que necesitan. Son los conocidos como métodos pasivos. Otros, en cambio, inyectan tráfico en la red en dirección al cliente para que éste lo reciba y evalúe. Son los métodos activos.

El objetivo de este TFC será el de diseñar una infraestructura viable para la gestión de la calidad de servicio que se debe ofrecer a cada uno de los usuarios de un servicio multimedia.

Para ello, en este primer capítulo se va a estudiar los parámetros que se utilizaran para determinar la calidad de servicio que se prestará al usuario. También se estudiaran y explicaran diversos métodos para obtener esta información. Se expondrán tecnologías para la comunicación de los elementos del sistema. Y finalmente se describirán las características que necesitará el servidor de contenidos, ofreciendo una posible solución para su implementación.

1.2. Definición del usuario: Parámetros de red

Existen varios parámetros que pueden servir para definir el nivel de calidad que ofrece una red para transmisiones de contenidos multimedia.

El **retardo** es el tiempo que tarda un paquete de datos entre el instante de su envío y el instante de su recepción, aunque en los sistemas de video en tiempo real también puede ser considerado el tiempo entre el instante en que se reproduce la vista en local y el instante en que se reproduce la vista en remoto.

El ***jitter***, también conocido como variación del retardo. Hace que sea necesario introducir un *buffer* de almacenamiento antes de la reproducción, ya que es posible que la información llegue pasado su instante de reproducción. Este *buffer* introduce un mayor retardo de reproducción, empeorando la sensación de presencia en los sistemas de video en tiempo real.

Las **pérdidas** introducen errores en la reproducción de los contenidos, así que son una molestia directa para el usuario. Además, con los nuevos sistemas de codificación de video por predicción de *frames*, una pérdida puede perdurar durante periodos relativamente largos desde el punto de vista del espectador.

Finalmente, el **ancho de banda** disponible es la cantidad de ancho de banda no utilizado en la red en un instante concreto. Se podría calcular como el ancho de banda teórico menos el tráfico ya existente en la red. Iniciar una transmisión a una tasa superior al ancho de banda disponible en el punto más restrictivo del camino producirá en el mejor de los casos un incremento progresivo del retardo, hasta acabar produciendo pérdidas tanto en la comunicación en curso como en el resto de las comunicaciones de la red.

Para reducir la complejidad que requeriría implementar un sistema que calculara y evaluara todos y cada uno de estos parámetros [8] [10] [21] [22], se ha decidido seleccionar aquellos que son más importantes. Dado que el envío a tasas mayores al ancho de banda disponible es una importante causa de pérdidas e incremento del retardo, se ha concluido que éste deberá ser el parámetro a analizar.

1.3. Obtención de parámetros por método pasivo: el capturador de paquetes

La implementación de un método pasivo requiere normalmente de un capturador de paquetes. El capturador de paquetes es un elemento que permite capturar los paquetes de datos que son enviados o llegan a una interfaz de red sin afectar a la comunicación a la que pertenecen los paquetes. De esta manera, se puede realizar tareas de monitorización, almacenar los datos capturados y analizar su contenido (Fig. 1.2).

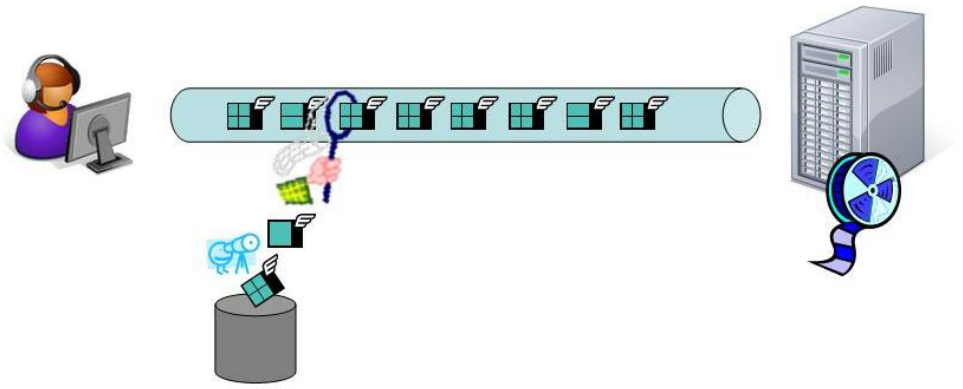

Fig. 1.2 Simplificación del proceso de captura de paquetes.

El capturador de paquetes utiliza la librería *pcap*, que proporciona una API para la captura de tráfico de la red. Para acceder a esta librería desde *Java* es necesaria una capa que realice la función de adaptador entre este lenguaje y la librería. Dos proyectos que realizan esta tarea son *JPcap* [15] [23] y *JnetPcap* [24], que proporcionan una API muy similar. También proporcionan soporte

para la identificación de ciertos protocolos de red y se encargan de su conversión a objetos *Java*. En este escenario se opta por una implementación con *JnetPcap* por proporcionar un soporte para un número mayor de protocolos, diferentes posibilidades de acceso a los paquetes capturados y soporte para almacenamiento de las capturas, únicamente con una dificultad de implementación ligeramente superior.

El capturador de paquetes (Fig. 1.3) se inicializa al darle una interfaz de red a la que escuchar, un tamaño máximo de paquete admisible, los *flags* que sean necesarios (como el de modo promiscuo, con el cuál se puede obtener todos los paquetes que lleguen a nuestra interfaz de red, aunque la máquina que los captura no sea su destinatario) y un tiempo máximo de espera. Una vez que se ha inicializado, se pueden aplicar filtros al capturador para determinar el tipo de tráfico que se desea monitorizar.

Finalmente se inicia la captura asignando al capturador un gestor, que se encargará de procesar los paquetes que pasen el filtro, e indicándole un número de paquetes a capturar (es posible configurarlo para que capture hasta que se le detenga). Los paquetes que llegan al gestor únicamente existirán mientras éste los esté procesando. Para dar persistencia a los datos será necesario que el gestor los almacene.

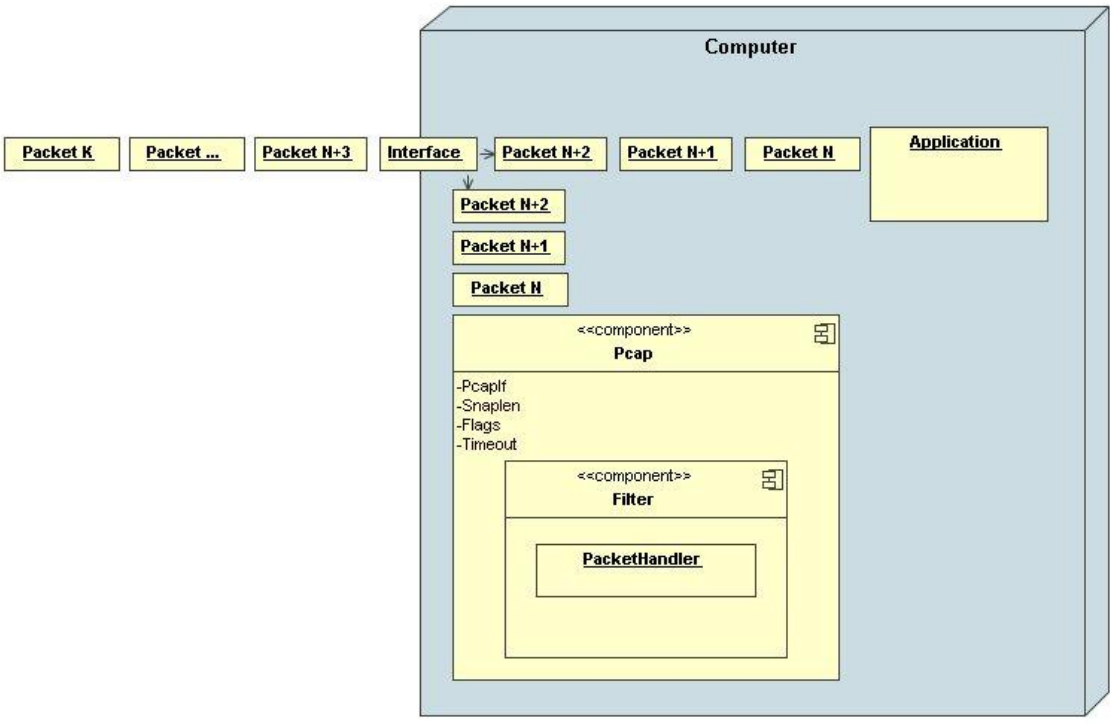

Fig. 1.3 Capturador de paquetes obteniendo el tráfico de la interfaz de red indicada.

Complementando este elemento con un generador de paquetes de implementación propia que representaría la fuente existente de los contenidos,

se han podido realizar medidas de pérdidas y *jitter* para el flujo de datos generado. Añadiendo un mecanismo de sincronización como el GPS sería posible determinar también el tiempo total desde la transmisión del paquete hasta su recepción y aproximar el ancho de banda disponible de la red [1]. Pero en un escenario de análisis pasivo, será necesario conocer el flujo que llevará los datos (direcciones IP, puertos, protocolos, tipo de contenido y formato) para poder utilizar la información de los protocolos de nivel de aplicación de manera que se puedan obtener los valores de las pérdidas y el *jitter* para flujos de media. Si se desea conocer también los tiempos de transmisión y el ancho de banda para flujos multimedia, sigue siendo necesario un mecanismo de sincronización.

1.4. Obtención de parámetros por método activo: técnicas de estimación de ancho de banda

Típicamente se utilizan dos definiciones para separar las técnicas de estimación de ancho de banda: PGM y PRM.

Las técnicas basadas en PGM envían parejas de paquetes con un intervalo determinado de tiempo entre ellos. Si el intervalo se mantiene, no hay tráfico cruzado. Si el intervalo crece, se calcula el nivel de tráfico cruzado y se resta al ancho de banda estimado.

Las técnicas basadas en PRM envían trenes de paquetes. Si la tasa de emisión es inferior al ancho de banda disponible mínimo del enlace, el flujo llegará a la misma tasa de manera más o menos constante. Si la tasa es mayor, los paquetes se irán acumulando en los *buffers* del enlace, añadiendo más retardo a cada paquete nuevo que llegue, y el flujo llegará cada vez a una tasa menor.

En entornos controlados, los métodos PGM parecen dar resultados más rápidos y acurados. Pero para entornos reales, con tráfico cruzado, el parámetro utilizado por estas técnicas para obtener los resultados (el tiempo entre paquetes) se ve afectado por los paquetes que se sitúan entre los paquetes de prueba y por los retrasos sufridos por tiempos de encolado. En cambio, como los métodos basados en PRM utilizan los retrasos producidos por la sobrecarga de los *buffers* como parámetro para resolver las peticiones, estos son más precisos a la hora de medir redes con tráfico cruzado. El aspecto negativo de estos métodos es que siguen siendo más lentos que los que se basan en PGR.

Packet Pair es de los primeros que se desarrollaron. La técnica consiste en enviar una pareja de paquetes iguales en un intervalo lo suficientemente pequeño de tiempo como para que se encolen juntos en todos los *buffers*. De esta manera, al llegar al *buffer* del enlace con menor capacidad, los paquetes sufren una separación por el mayor tiempo de procesado del primero de los paquetes, que hace que el segundo tarde más en empezar a ser procesado y salir (Fig. 1.4). Se clasifica como PRM porque, aunque el tren sea de dos paquetes, la tasa final para flujos de más tasa que el ancho de banda

disponible mínimo acaba reduciéndose al incrementar el tiempo entre la llegada de los paquetes.

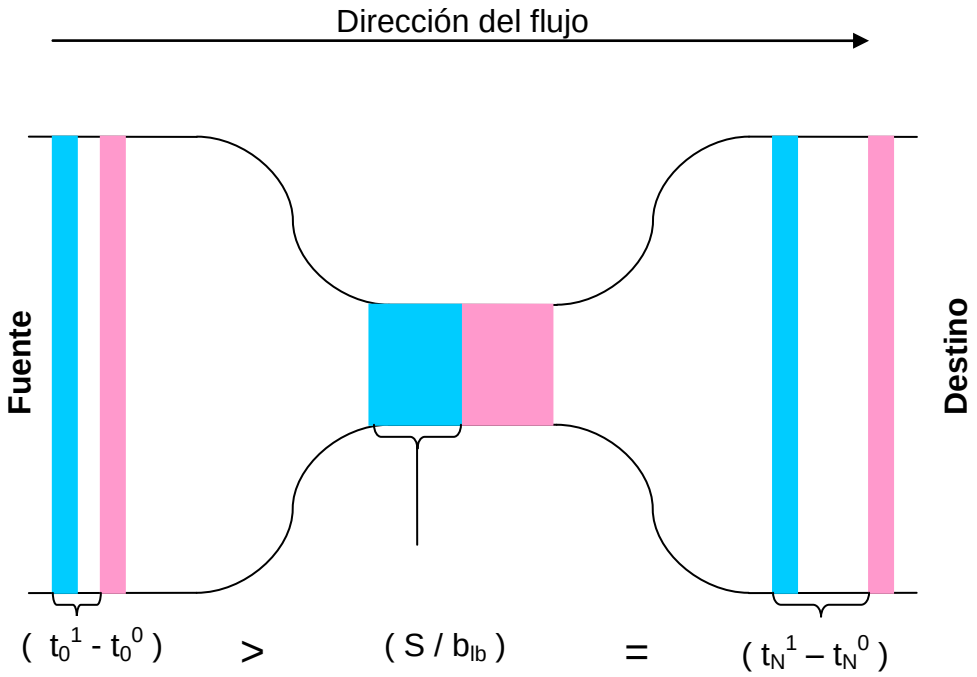

Fig. 1.4 Incremento del tiempo entre paquetes por encolado en cuello de botella.

La técnica de estimación de ancho de banda por *Packet Pair* se basa en la siguiente premisa:

"Si dos paquetes se envían lo suficientemente juntos como para que cuando lleguen a un *buffer* siempre se encolen uno junto al otro, la diferencia entre los instantes de recepción de ambos paquetes es igual al valor máximo entre la diferencia entre los instantes de transmisión o el ancho de banda entre la longitud del paquete"

$$t^1_n - t^0_n = \max\left(\frac{s}{b_l}, t^1_0 - t^0_0\right) \qquad \textbf{(1.1)}$$

Por tanto, en principio es posible estimar el ancho de banda en el cuello de botella de la comunicación. Además, tiene en cuenta la posibilidad de que el cuello de botella sea la propia fuente, evitando estimaciones incorrectas para diferencias entre instantes de recepción menores a las diferencias entre instantes de transmisión (por efecto de la red).

En las pruebas que se han realizado, se ha utilizado una fuente UDP que genera trenes de paquetes entre los que, se va midiendo la diferencia entre los instantes de llegada. Luego, se filtran los resultados para poder analizarlos y obtener el ancho de banda.

Esta técnica tiene una serie de problemas por resolver [4]. Por un lado, es muy difícil que se envíen dos paquetes tan extremadamente juntos sin utilizar una fuente específicamente diseñada para ello. Se tiene que confiar en que los *buffers* no introducirán paquetes cruzados entre los de prueba (son necesarias colas FIFO). Además, el ancho de banda estimado es el de transmisión del cuello de botella. No se tiene en cuenta el resto del tráfico de la red. Realizar un envío de tráfico a esa tasa iría llenando los *buffers*. La velocidad a la que se llenarían dependería del nivel de tráfico cruzado existente, pero para un envío sostenido acabaría desbordando y perdiendo paquetes. En el caso concreto de las comunicaciones multimedia, el retraso haría que el tiempo entre paquetes fuera aumentando, de manera que la reproducción iría más lenta o sufriría cortes para llenar el *buffer* de reproducción. Finalmente, cuando se empezaran a producir pérdidas, la calidad decaería drásticamente.

La utilización de esta técnica puede ser válida en laboratorios dedicados, pero existen muchos problemas para su utilización en entornos reales. Además, el valor resultante de la medida no resulta útil ya que se desconoce el nivel de tráfico cruzado de las redes que son atravesadas durante la comunicación.

Ante esa barrera, se empezaron a desarrollar técnicas más resistentes gracias a la utilización de trenes de paquetes más largos y a la utilización de la idea del decremento del ancho de banda. Una de las primeras y más simples que se pueden encontrar es *Packet Triplet*, que ya apunta a la observación de varios intervalos entre paquetes para mejorar la resistencia, o *PTR*, que utiliza trenes de paquetes.

Llegados al momento de seleccionar un algoritmo, parece ser que hay varios que tienen una precisión similar [5] [6]. Por cantidad de información y soporte para la implementación se ha escogido *Pathload* como método de estimación.

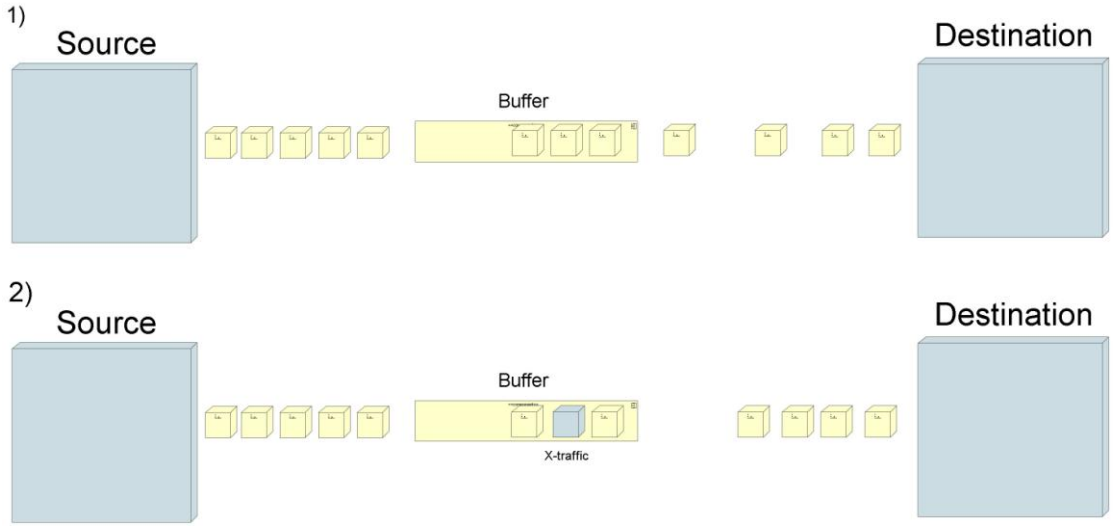

Fig. 1.5 Trenes de paquetes a tasa superior 1) e inferior 2) al ancho de banda disponible mínimo

Pathload [3], a diferencia de *Packet Pair*, no mide la tasa a la que envía el cuello de botella de la comunicación, sino el mínimo ancho de banda disponible. Se podría definir el ancho de banda disponible como el ancho de banda restante si se resta al ancho de banda real el ancho de banda ocupado por el tráfico cruzado de la red en ese punto. Por tanto, el elemento que se mide con esta técnica no tiene por qué ser el que tenga una tasa de transferencia más baja, sino aquel que disponga de menos ancho de banda libre.

El método utilizado por *Pathload* para medir ese valor consiste en asumir que, si se envía un flujo de datos a través de la red a una tasa superior a la del mínimo ancho de banda disponible, los *buffers* de la red irán acumulando paquetes. Por tanto, el tiempo que tardaran en llegar los paquetes de la fuente al destino ira en incremento hasta que los paquetes acabaran por perderse. En cambio, si la tasa es menor que ese umbral, los paquetes tardaran más o menos lo mismo en llegar desde la fuente al destino.

Así, si se realizan pruebas sucesivas para valores crecientes o decrecientes de ancho de banda de transmisión, es posible establecer un margen en el que se debería encontrar el ancho de banda disponible mínimo del camino utilizado. Cada prueba deberá ser suficientemente larga como para poder apreciar el crecimiento del tiempo entre el envío y la recepción y se tendrá que repetir varias veces para poder comprobar que se trata de algo representativo, y no de una situación puntual.

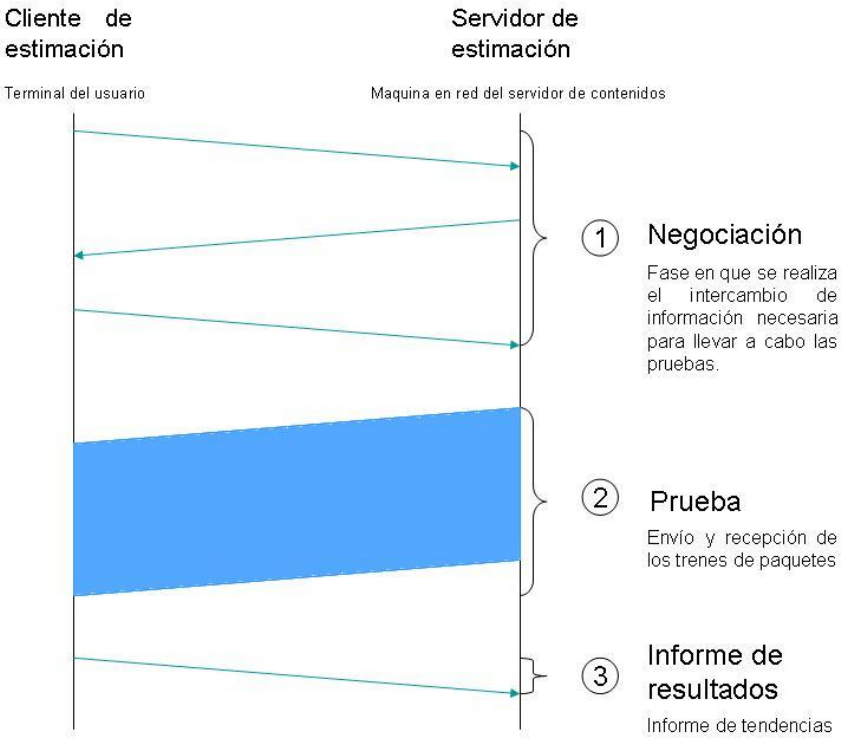

Fig. 1.6 Simplificación de un ciclo del proceso de estimación

En la representación simplificada de un ciclo de pruebas (Fig 1.6) se pueden observar tres fases claramente diferenciadas. En la fase de negociación, se decide que se va a realizar una prueba y las características que tendrá (tasa de envío, tamaño de los paquetes y tiempo entre envío de paquetes). La fase de prueba consiste en transmitir los trenes de paquetes desde el servidor hasta el cliente. El cliente analiza los datos de las pruebas y pasa a la fase de informe de resultados, donde se indica al servidor de la tendencia - creciente o no creciente - que han mostrado los retardos de los paquetes.

Se deben establecer valores de resolución adecuados para la diferencia entre márgenes, o la aplicación tardará un tiempo excesivo en resolver la petición de estimación. Además, hay que tener en cuenta que, para tasas de prueba situadas alrededor del valor real de ancho de banda disponible, es muy probable que haya una zona de indeterminación, en la que no es posible asegurar si el ancho de banda crece o decrece.

El esquema (Fig. 1.5) muestra como al enviar flujos de paquetes a tasas superiores a la del ancho de banda disponible - caso 1 - los propios paquetes del flujo van acumulándose en los *buffers* (que aquí está representado como uno solo) que hay entre el origen y el destino de los paquetes. Eso hace que cada paquete tenga que esperar un tiempo mayor a ser procesado y llegue más tarde al destino. En cambio, en el caso en que la tasa de envío es inferior al ancho de banda disponible, los paquetes acaban tardando más o menos lo mismo en llegar, siendo afectados únicamente por las variaciones en el tráfico cruzado.

Con la implementación realizada de *Pathload*, se ha estimado el ancho de banda para enlaces a 10 y 100 Mbps, dando resultados alrededor de un 85% del valor teórico en un tiempo inferior a 20 segundos y a pesar del tráfico cruzado existente (Fig. 2.7).

1.5. Comunicación entre los elementos

Como se ha mostrado, para poder llevar a cabo los objetivos de este proyecto es necesaria la utilización de múltiples elementos, que se tendrán que situar en múltiples máquinas. Esto significa que deberán comunicarse para coordinar la ejecución de los diversos servicios.

La comunicación se puede hacer a través de implementaciones propias sobre los protocolos existentes o se puede delegar esta tarea a tecnologías propietarias que ya han sido diseñados previamente con este objetivo.

Un grupo de tecnologías que pueden cumplir con estas tareas de coordinación son las tecnologías de gestión remota. La gestión remota permite administrar aplicaciones o dispositivos desde una aplicación externa. De esta manera, se

puede dar instrucciones a estos elementos y monitorizar sistemas completos sin necesidad de tener que estar directamente conectado al dispositivo o la aplicación.

Hay diversas tecnologías que permiten realizar gestión remota. Entre ellas se encuentran SNMP o las series de productos Tivoli de IBM, Microsoft Operations Manager y HP Openview, entre muchos otros.

JMX es una tecnología que proporciona herramientas para realizar gestión remota [2]. Añadiendo una capa de instrumentación, da la posibilidad de hacer aptas para la gestión remota aplicaciones que antes no lo eran. Gracias a este hecho, es posible construir una estructura virtual en que cada elemento está representado por una *MBean* (un objeto de *java* que tiene como objetivo dar acceso a atributos y servicios de los elementos que contiene).

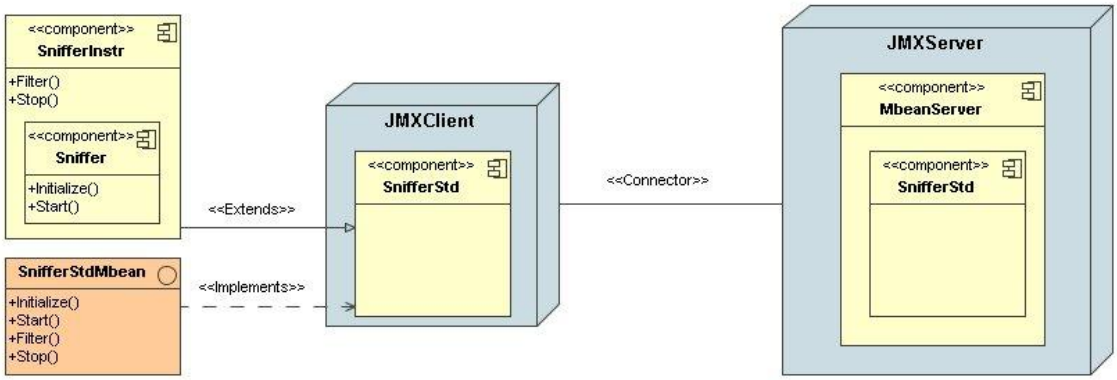

Fig. 1.7 Cliente y servidor JMX. El cliente crea una *MBean* que hace la instrumentación de un capturador de paquetes y la registra en un servidor de *MBeans*.

Para probar la tecnología se ha diseñado [14] un capturador de paquetes escrito en *java* gestionado remotamente (Fig. 1.7). La instrumentación se realiza creando una clase de *java* que herede de la de la aplicación original y añadiendo ahí las funciones que sean necesarias para cumplir con la definición de la interfaz de la *MBean*. La interfaz de la *MBean* describe los métodos que serán accesibles para la gestión remota. El objeto final que será gestionado extenderá de la clase de instrumentación e implementará la interfaz de la *MBean* para que el servidor de *MBeans* sea capaz de trabajar con él. Una ventaja de este sistema es que permite utilizar diferentes implementaciones de instrumentación sin modificar el resto del sistema. Una vez instanciado este objeto, tendrá que registrarse en el servidor de *MBeans*.

El servidor de *MBeans* es el elemento central desde el que se realiza la gestión remota. Este elemento almacena *MBeans* creadas en el propio servidor, así como representaciones de *MBeans* remotas. Su función es dar información sobre las *MBeans* que tiene registradas y sobre sus servicios, dar acceso a dichos servicios y ofrecer servicios propios (ya sea para la gestión de *MBeans* o para cualquier otra tarea).

La comunicación con el servidor de *MBeans* se realiza a través de un conector. Este elemento se encarga de transmitir las peticiones al servidor y de comunicar las *MBeans* registradas remotamente con el servidor de *MBeans*. Actualmente la única implementación del conector utiliza RMI, pero ya hay proyectos en desarrollo de conectores utilizando Servicios Web, como el JSR 262 [18] [19] [20].

Se han realizado pruebas de gestión con JMX, utilizando tanto el conector RMI como el *Early Access* 4 de JSR 262. La gestión con conector RMI ha funcionado correctamente, permitiendo la inicialización del capturador, la captura del tráfico indicado en el filtro y la detención de la captura. El aspecto negativo de esta prueba es que requiere de un puerto a fijar para poder realizar la comunicación por RMI. Esto presenta problemas a la hora de entrar en redes con control de tráfico, que pueden identificar el flujo de datos como no deseado y bloquear su acceso a la red al no ser un protocolo estándar con puerto asignado, como ocurre con HTTP. La gestión con el conector por Servicio Web no ha funcionado como se esperaba. A pesar de que en teoría los conectores deberían poder intercambiarse de manera transparente, el sistema no ha funcionado al realizar el cambio. Es posible que se deba a que no se trata de una versión definitiva del producto y por tanto todavía existen problemas internos a solucionar.

1.6. Servidor de contenidos y transcodificación

Como ya se ha comentado, un sistema de video bajo de manda tiene un elemento que se encarga de enviar el video al dispositivo del cliente, el servidor de contenido. Pero para poder proporcionar calidad de servicio que se quiere ofrecer se debe hacer que este elemento realice tareas de transcodificación.

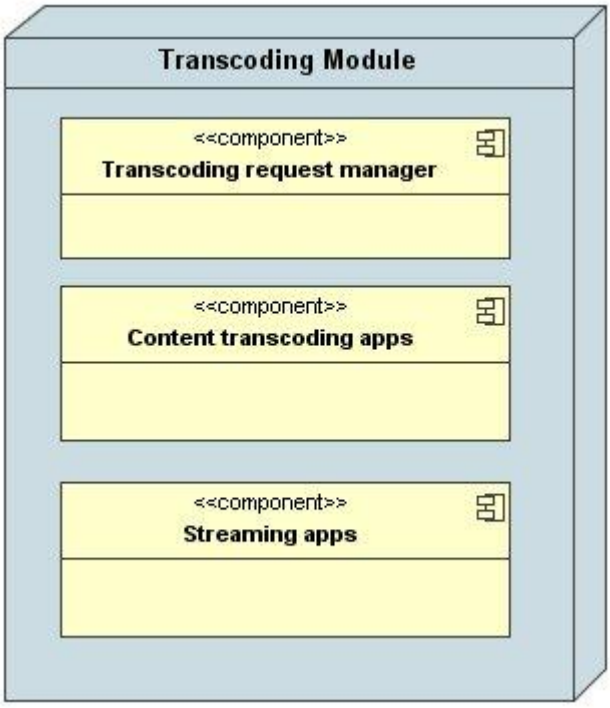

Fig. 1.8 Componentes del gestor de peticiones de transcodificación

Por definición, una transcodificación consiste en cambiar el *codec* de un elemento de media. Pero también puede ser considerada como transcodificación el proceso de cambiar cualquier característica de un elemento de media, ya sea su tamaño de almacenamiento, su tasa de reproducción, su anchura o altura, etc.

Alembik es un servidor de transcodificación de media. El núcleo de este servidor es el gestor de peticiones de transcodificación (Fig. 1.8). Este elemento no realiza ningún tipo de transcodificación, sino que se encarga de recibir las peticiones, dar instrucciones a las aplicaciones asociadas para que realicen una determinada tarea y enviar una respuesta a la petición con información sobre el resultado.

Las aplicaciones de transcodificación asociadas a *Alembik* son *ffmpeg* para la transcodificación de audio y video e *ImageMagick* para la transcodificacion de imágenes.

Alembik también permite realizar peticiones para transmisión de media en *streaming*. Para ello utiliza las aplicaciones *MP4Box* y *Darwin Streaming Server*.

Las peticiones pueden realizarse a través los protocolos HTTP, RMI y SOAP y pueden ser síncronas o asíncronas. Por ello, *Alembik* no solo acepta peticiones de transcodificación, sino que también permite consultas de estado para tareas en curso.

Hay dos tipos de peticiones de transcodificación. La primera consiste en indicar a *Alembik* el tipo de aplicación o dispositivo para el que se va a adaptar la media. Este tipo de peticiones se resuelven gracias a la utilización por parte de *Alembik* de un fichero WURFL, en el que se listan las características de una gran cantidad de dispositivos. La segunda consiste en proporcionar un identificador de perfil para un perfil previamente definido. Estos perfiles se almacenan en un fichero profile-defs.xml en el mismo directorio en el que se encuentra el fichero WURFL.

Si no se proporciona esta información a *Alembik*, intentará obtenerla del mensaje en el que venia la petición (por ejemplo, del campo *UserAgent* de las peticione HTTP). Si aun así es incapaz de encontrar un perfil que le de información sobre cómo se debe realizar la transcodificación de la media, *Alembik* no tendrá datos suficientes para poder llevar a cabo el proceso y se responderá a la petición con un mensaje de error.

Con todo esto *Alembik* se convierte en una interesante opción para ser utilizado como servidor de contenidos y transcodificador de video a la vez.

Capítulo 2. IMPLEMENTACIÓN

2.1. Entorno de trabajo

Este TFC se ha desarrollado en las instalaciones que la fundación i2CAT posee en la EPSC.

El material que se ha utilizado ha consistido básicamente en dos máquinas de uso compartido con otros proyectos. Para solucionar el problema de la movilidad entre máquinas, se ha obtenido acceso a un sistema de control y almacenamiento de versiones *Subversion*. Y en ocasiones puntuales, si se ha requerido de un mayor número de máquinas para poder realizar pruebas sobre algún escenario, también se ha tenido acceso a ellas.

Una de las máquinas utiliza sistema operativo *Windows XP*. Su principal cometido ha sido funcionar como máquina de desarrollo y testeo para las aplicaciones de software que se han ido implementando. A la vez, ha sido utilizada para la búsqueda de información y la redacción de documentos como esta misma memoria.

La otra máquina dispone de una configuración con dos particiones. En una, utiliza sistema operativo *Windows XP* y ha servido para llevar a cabo las mismas tareas que se realizaban en la primera máquina en los espacios de tiempo en que estaba ocupada para otros proyectos. También se ha utilizado en conjunción con la primera para probar las aplicaciones que requirieran de comunicación entre máquinas remotas.

La otra partición de la segunda máquina utiliza un sistema operativo basado en *Linux, Ubuntu*. Básicamente, ha estado funcionando como máquina de implementación para el servicio de gestión de peticiones de transcodificación. También ha resultado útil para el testeo de las aplicaciones que después serian utilizadas de forma remota, dado que en sistemas operativos *Windows* no es posible analizar el tráfico enviado a la interfaz de *loopback* con la aplicación que se utiliza para ese fin, *Wireshark*. El cambio de sistema operativo es viable porque el software utilizado es multiplataforma y las aplicaciones desarrolladas se han implementado en *Java*. Esas pruebas locales realizaban la misma función que la eliminación de una variable - en este caso, la variable “enrutamiento, envío y recepción de mensajes” - en una función. De esta manera, los únicos elementos restantes eran las lógicas de negocio de los cliente y servidores del escenario que se estuviera probando y resultaba más sencillo localizar la fuente de los errores.

El control sobre las múltiples máquinas utilizadas en el desarrollo del proyecto se ha ejercido normalmente desde una única máquina, utilizando tecnologías de escritorio remoto como VNC o de consola remota como SSH. La integración de estas tecnologías durante el desarrollo del proyecto ha minimizado la pérdida de tiempo de trabajo por desplazamientos y el desgaste físico del

desarrollador por repetición de movimientos (levantarse y sentarse repetidamente, giros continuos para trabajar entre máquinas contiguas, etc.).

El servidor JEE utilizado para el despliegue de los WS es Apache Tomcat 6. Los WS se han desarrollado utilizando el paquete *JAX-WS* en su versión 1.5.

Con el objetivo de poder utilizar la aplicación en el mayor número de máquinas posible, se ha escogido *Java* como lenguaje de programación. Para el desarrollo en *Java* es necesario un JDK. La versión utilizada en este proyecto es la 6 de *Sun Microsystems*. Del mismo modo, para poder ejecutar las aplicaciones *Java* es necesaria una máquina virtual *Java* o JRE. El JRE utilizado es el que viene incluido con el JDK.

El software utilizado para la redacción de la memoria es Microsoft Office 2003 / 2007 y *MagicDraw* 11 para el diseño de esquemas.

2.2. Tecnologías y herramientas utilizadas

- *JDK 6 SE*: Versión de desarrollo de *Java*, que además proporciona el entorno de ejecución. Utilizado para desarrollar el software de la implementación, los Servicios Web, y ejecutar las herramientas necesarias.
- *Apache Tomcat 6*: Servidor de aplicaciones JEE que permite, entre otras cosas, el despliegue de Servicios Web.
- *JAX-WS*: Kit de desarrollo en *Java* que permite crear Servicios Web que pueden ser desplegados dentro del propio servidor de aplicaciones *Tomcat*.

2.3. Módulo de estimación de ancho de banda

El módulo de estimación de ancho de banda se crea con la intención de conseguir un elemento que pueda proporcionar información sobre la red del cliente. A través de una serie de pruebas previas al envío de contenido, el módulo es capaz de determinar el ancho de banda disponible del usuario. El ancho de banda disponible no es la capacidad del enlace del usuario hacia la red, sino el ancho de banda restante si se resta el ancho de banda en uso a la capacidad del enlace. De esta manera, el módulo de estimación de ancho de banda puede informar al elemento del sistema encargado de enviar los contenidos sobre cuál es la tasa optima para transmitir el contenido para ese usuario.

El usuario no mejorará la visualización del contenido si la tasa supera ese valor, porque su propia red limitará la tasa de transferencia y, lo que es peor, introducirá pérdidas. En cambio, cualquier transmisión a una tasa inferior a la que el usuario es capaz de recibir implica una pérdida de calidad sin justificación.

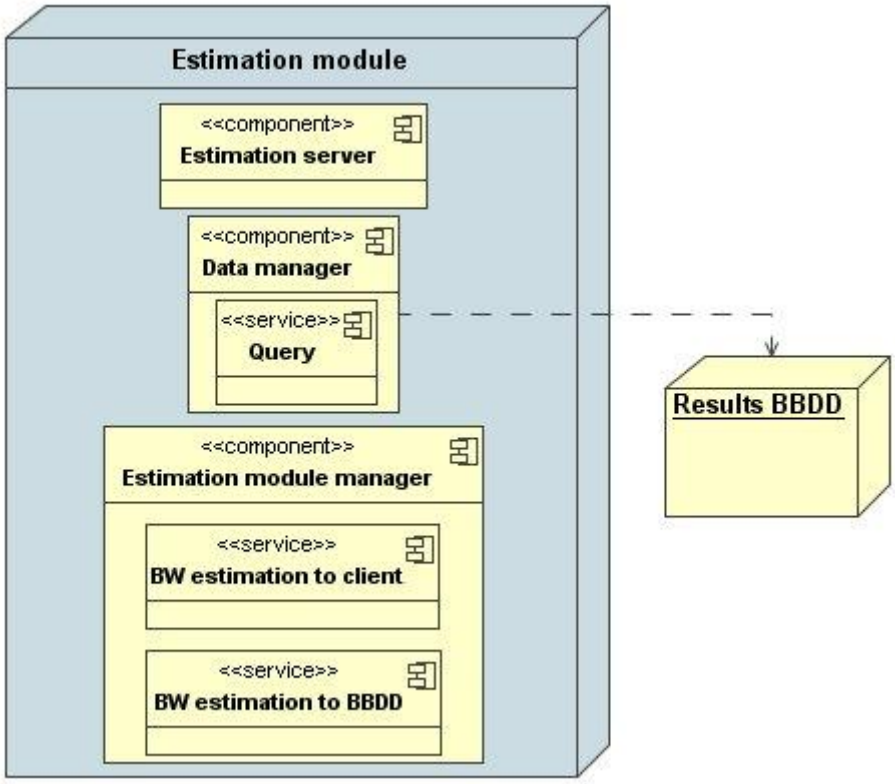

Fig. 2.1 Componentes del módulo de estimación de ancho de banda

El componente que forma el núcleo del módulo de estimación de ancho de banda es la parte servidor de la aplicación que implementa el método de estimación (Fig. 2.1). En combinación con la parte cliente de la aplicación, realiza una serie de pruebas que permiten acotar progresivamente el valor real del ancho de banda disponible.

El gestor de datos es un elemento de soporte para dar persistencia a las estimaciones ya realizadas. Estos datos podrían ser utilizados tanto para la validación de resultados como para poder ofrecer una respuesta a quien pudiera realizar una petición para determinar el ancho de banda de un usuario contra el que no se puede establecer una conexión para hacer las pruebas.

El gestor del módulo de estimación funciona como intermediario entre el módulo de estimación de contenidos y aquellos que realizan las peticiones. De esta manera es posible modificar el funcionamiento del módulo sin tener que realizar cambios en los elementos que acceden a sus servicios. El único elemento que está implementado para poder realizar peticiones al módulo de estimación de ancho de banda es el cliente de la aplicación que estima el ancho de banda. Este acceso se da cuando quien accede al módulo es el propio usuario para realizar una consulta rápida sobre cuál es su ancho de banda disponible.

2.3.1. Implementación de la técnica de estimación *Pathload*

Pathload es una técnica de estimación de ancho de banda que se basa en la observación del retraso en un único sentido para determinar el ancho de banda disponible para ese sentido de la comunicación.

El procedimiento para la estimación consiste en una serie de pruebas. Durante cada prueba, se repite un proceso en el que se envían trenes de paquetes a una tasa y de un tamaño conocidos hacia un cliente. El cliente observa la variación del retardo desde el instante del envío al momento de la recepción.

Si la tendencia de la medida del tren de paquetes es que el retardo crezca, entonces los *buffers* del camino se han ido saturando por no poder procesar esa tasa de información con el ancho de banda disponible del que disponían antes de empezar la prueba. En cambio, si los resultados tienden a ser iguales, la tasa de envío utilizada estará por debajo del ancho de banda disponible mínimo en el camino.

Tras procesar una cantidad concreta de trenes de paquetes, el cliente informa al servidor sobre la tendencia mayoritaria del grupo de pruebas y este último decide una nueva tasa para realizar la siguiente prueba.

Este procedimiento finaliza cuando el servidor decide que la precisión del resultado es suficiente.

2.3.2.1 Mensajes

La implementación realizada utiliza una capa de comunicación entre los clientes y la parte servidor de la aplicación. Esta capa se ha implementado sobre el protocolo TCP con un conjunto de mensajes desarrollados exclusivamente para esta aplicación.

La implementación tiene un funcionamiento activo, lo que significa que introduce tráfico en la red para realizar las pruebas. Para ello utiliza paquetes UDP con un formato de datos que tiene por objetivo proporcionar al cliente la información que necesita para analizar el tren de paquetes. Además, la fuente de paquetes ha sido diseñada específicamente para la emisión de paquetes en intervalos muy pequeños de tiempo.

El intercambio de mensajes para una comunicación sin errores se muestra en el siguiente diagrama de flujo (Fig. 2.2) y es el siguiente:

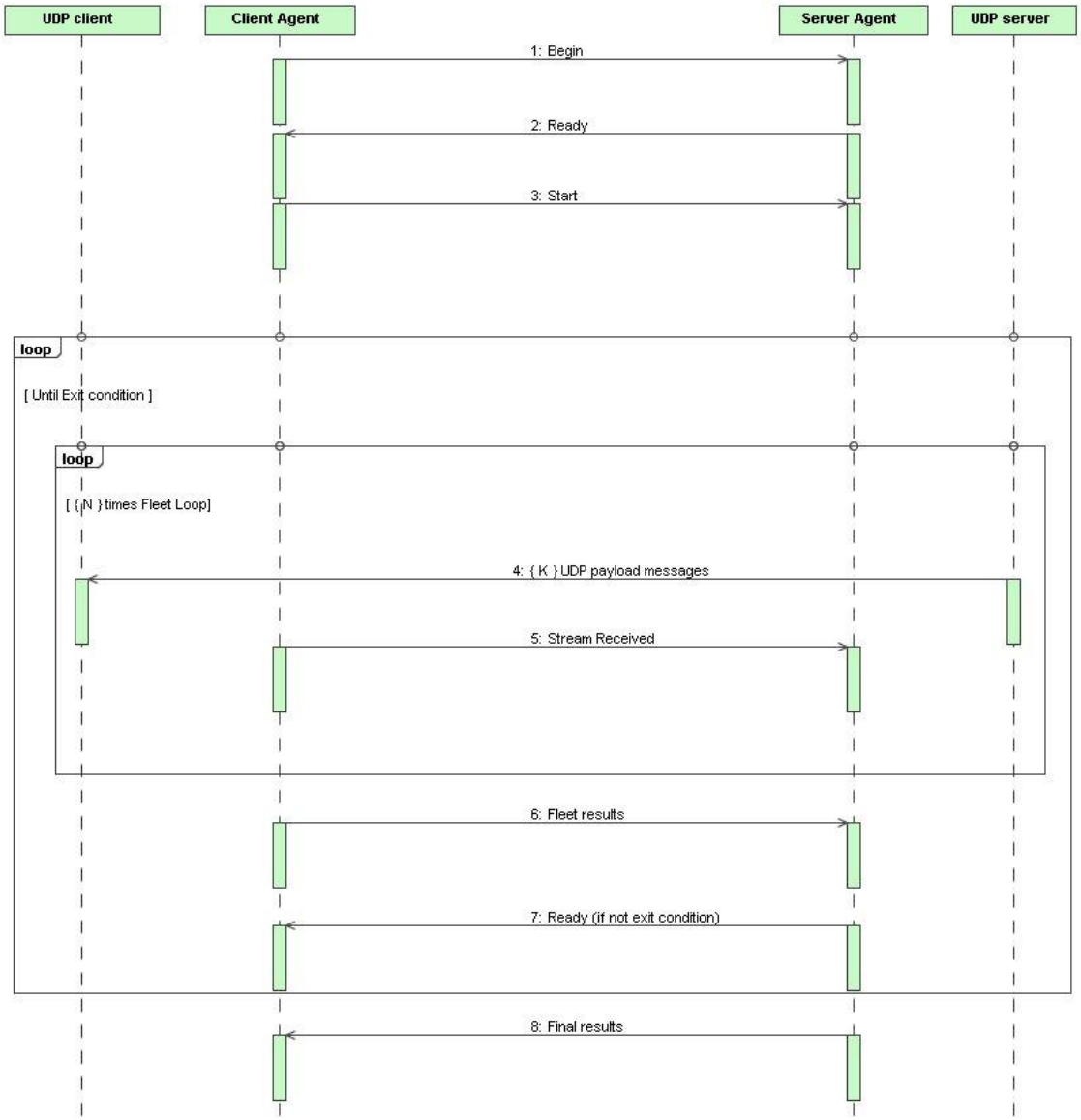

Fig. 2.2 Diagrama de secuencia del intercambio de mensajes para la estimación de ancho de banda

Se intercambian los datos necesarios para el correcto funcionamiento de la aplicación (1 y 2) y el servidor informa de que está listo para iniciar el primer envío (3).

Se entra en el primer bucle hasta que se cumpla alguna de las condiciones de salida, donde las condiciones de salida para un funcionamiento sin errores son: a) que la aplicación ha obtenido unos valores de ancho de banda dentro de lo establecido por el usuario o b) que los valores máximos y mínimos estimados están dentro de los limites de proximidad definidos por el usuario a la zona de indeterminación (no se puede precisar más el resultado).

Se entra en el segundo bucle, en el que se repite el envío (4), confirmación (5) y análisis de los trenes (*streams*) UDP mientras que el número de trenes confirmados sea inferior al número de trenes que se ha acordado enviar.

Se informa de los resultados obtenidos tras estudiar los análisis realizados previamente (6). Si todavía no se cumple ninguna de las condiciones de salida, se vuelve a informar de que el servidor está listo para otro envío.

Si se cumple alguna de las condiciones de salida, se informa al cliente del resultado de la estimación.

Los mensajes para esta comunicación sin errores són:

- Begin:
  - Función: Cliente indica al servidor que quiere iniciar el proceso de estimación y le ofrece ciertos parámetros seleccionados por el usuario.
  - Parametros:
    - Obligatorios: Puerto UDP, #Opciones
    - Opcionales: Ancho de banda inicial, número de paquetes por tren (K) y número de trenes por prueba (*fleet*) (N).
- Ready:
  - Función: Servidor indica a cliente los valores finales de las variables que necesita.
  - Parámetros:
    - Obligatorios: Número de paquetes por tren, tamaño de paquete, tiempo entre paquetes y número de trenes por prueba.
    - Opcionales: -
- Start:
  - Función: Cliente informa al servidor que está listo para recibir los paquetes del tren de prueba.
  - Parámetros:
    - Obligatorios: -
    - Opcionales: -
- { K } UDP Payload Messages:
  - Función: Tren UDP formado por K paquetes de longitud L enviados a una tasa R=L/T.
  - Parámetros:
    - Obligatorios: Identificador de prueba, Identificador de tren, Identificador de paquete, *timestamp* y *padding*.
    - Opcionales: -
- Stream Received:
  - Función: Cliente informa al servidor de que puede contabilizar ese tren como correctamente recibido.
  - Parámetros:
    - Obligatorios: -
    - Opcionales: -
- Fleet Result:
  - Función: Cliente ha estimado la tendencia de la prueba e informa del resultado al servidor.
  - Parámetros:

        - Obligatorios: Tendencia.
        - Opcionales: -
- Final Result
    - Función: Servidor informa al cliente del resultado de la estimación.
    - Parámetros:
        - Obligatorios: Ancho de banda máximo estimado, Ancho de banda mínimo estimado, #Opciones.
        - Opcionales: Proceso abortado.

Los errores en el proceso de estimación se pueden producir por pérdidas de los paquetes UDP o por cancelación de la operación. Los mensajes a utilizar en estas situaciones son:

- Stop Fleet:
    - Función: Cliente pide a servidor que detenga el envío de a prueba por detección de pérdidas.
    - Parámetros:
        - Obligatorios: -
        - Opcionales: -
- Cancel:
    - Función: Cliente ordena al servidor que cancele todo el proceso por petición del usuario cliente.
    - Parámetros:
        - Obligatorios: -
        - Opcionales: -

2.3.2.2 Procesado de resultados

La implementación distribuye el procesado de los datos entre el cliente y el servidor. Esto se hace tanto por liberar de algo de carga al punto central del sistema de estimación, como por optimización de tiempo y recursos. Enviar los datos que todos los clientes almacenan sobre los paquetes recibidos al servidor tras cada envío de un tren de paquetes para que éste los procese, cuando el tratamiento de esos datos es una operación bastante simple que no requiere de un gran procesado por parte de los clientes, es menos eficiente que dejar que los clientes procesen los datos e informen únicamente del resultado final de cada prueba.

Hay tres procesados de datos, donde cada uno utiliza los resultados del anterior de forma cíclica: El procesado de los trenes de paquetes para determinar su tendencia, el procesado de las tendencias de los múltiples trenes de una prueba para determinar si el retardo es creciente o no, y el cálculo de una nueva tasa de envío a partir del resultado creciente o no creciente de las pruebas anteriores para enviar los siguientes trenes de paquetes.

Previamente al procesado de los trenes de paquetes, se observan los niveles de pérdidas. Para altas tasas de pérdidas, o tras la acumulación de varios trenes de datos con niveles de pérdidas a tener en cuenta, se envía una advertencia al servidor. En el algoritmo original, estas situaciones implicaban la

repetición de la prueba. En la implementación que se ha hecho, se utiliza esos avisos como indicativos del estado de la red, de manera que si hay muchas pérdidas se considera que la red está sobrecargada y la tasa es demasiado elevada.

En el procesado de los trenes de paquetes, se toma como datos los instantes de salida y llegada de los paquetes y se introducen en una serie de ecuaciones matemáticas para determinar - según parámetros empíricos – si la tendencia del retardo es creciente, no es creciente o si no puede ser determinada.

El procesado de las tendencias se realiza una vez que se han procesado todos los trenes de paquetes. Ésta es una simple decisión por mayoría, en la que si hay una gran mayoría de resultados que indican una misma tendencia, se proporciona como resultado. Si no hay suficientes resultados con la misma tendencia para ninguna de las dos, el proceso se resuelve respondiendo que la tendencia es indeterminada. Como se ha comentado antes, cuando se detectan altos niveles de pérdidas en un tren o múltiples detecciones de pérdidas significativas en una prueba, la prueba se detiene y se considera que la tasa de envío supera de largo el ancho de banda disponible.

El procesado de los paquetes y las pruebas se realiza en los clientes. El nivel de pérdidas se calcula en el cliente, pero se toman acciones para solucionarlo en ambos bandos.

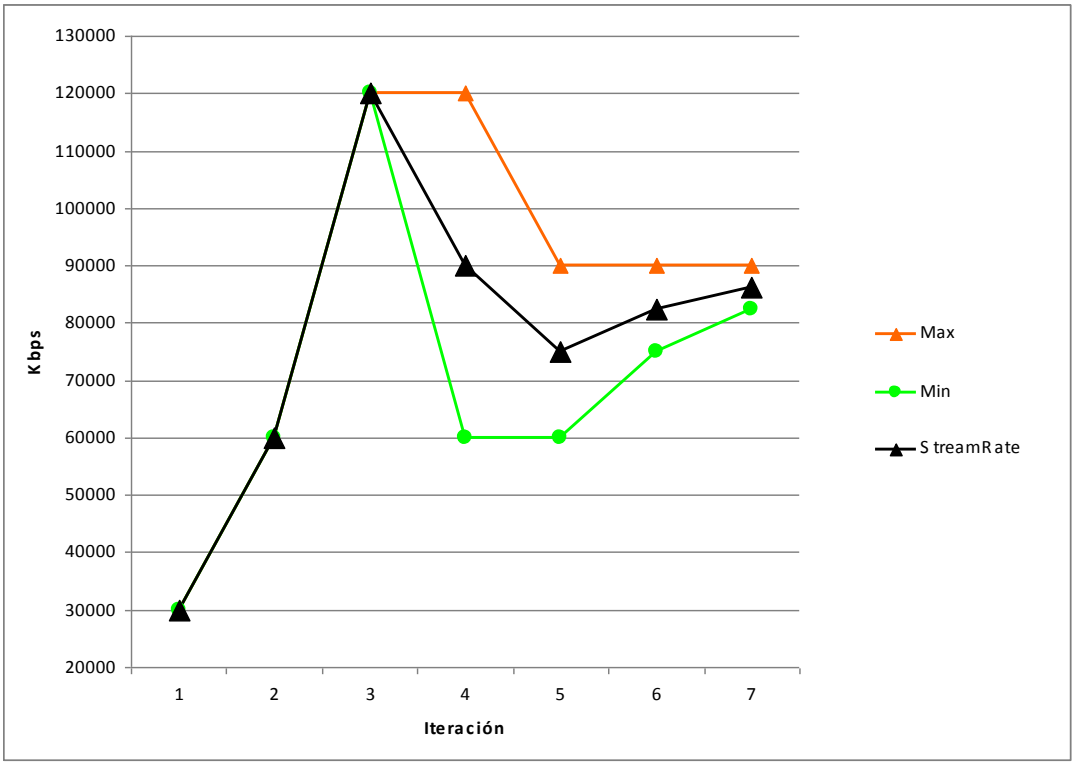

Fig. 2.3 Valores para la tasa de envío, margen máximo y margen mínimo durante un proceso de cálculo de la tasa de envío sin tendencias indeterminadas

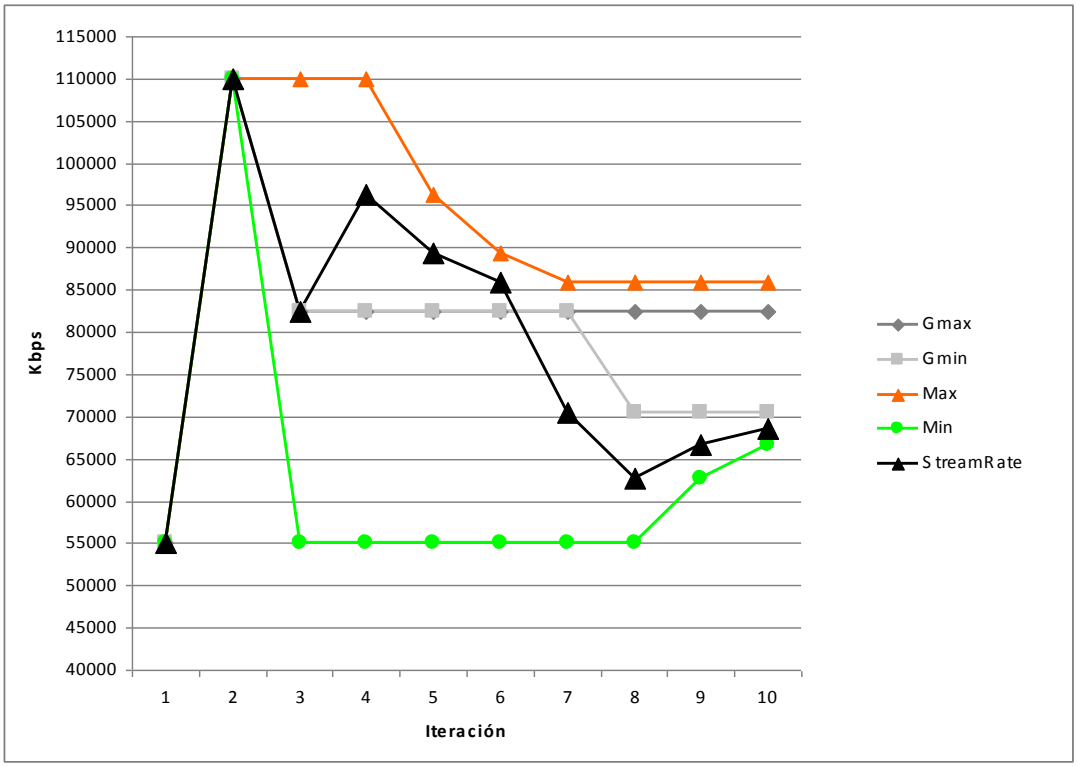

Fig. 2.4 Valores para la tasa de envío, márgenes máximo y mínimo, y márgenes máximo y mínimo de indeterminación durante un proceso de cálculo de la tasa de envío con tendencias indeterminadas

Conociendo ya la tendencia de la prueba, se calcula un nuevo valor para la tasa de envío, que intentará acercarse más al valor del ancho de banda disponible. El algoritmo utilizado tiene en cuenta los valores máximos y mínimos tanto de las tasas cuyas tendencias se han podido verificar, como de las que no han podido ser determinadas (Fig. 2.3 y Fig. 2.4). Se ha modificado ligeramente el algoritmo para obtener resultados algo más rápidamente. El objetivo es obtener la tasa máxima a la que se puede enviar sin saturar los *buffers*, y esa tasa es la tasa mínima estimada. Por tanto, una vez que ese valor se encuentre dentro de los márgenes establecidos, la aplicación lo tomará como una condición de salida y finalizará sin necesidad de ajustar la tasa máxima.

El cálculo de la nueva tasa se realiza en el servidor.

2.3.2.3 Generación y recepción de trenes de paquetes

Para el procesado de los trenes de paquetes, se requiere primero de un correcto envío de los paquetes que componen los trenes. Los trenes de paquetes se deben enviar a una tasa previamente determinada. Se utilizan dos variables para fijar esta tasa: el tamaño de los paquetes y el tiempo entre el envío de un paquete y el envío del siguiente.

La documentación consultada recomienda un tamaño mínimo y máximo de paquete y un tiempo mínimo entre el envío de un paquete y el siguiente. Este tiempo se fija para procurar evitar que sea la propia máquina receptora la que vaya encolando los paquetes en su *buffer* de entrada y altere así los resultados. Una tasa demasiado elevada de paquetes puede hacer que se acumulen en la máquina receptora mientras que ésta recibe y procesa los paquetes previos. El tiempo mínimo entre paquetes es de un centenar de microsegundos, pero si el tamaño del paquete se acerca al límite, será menor.

Para cumplir con valores tan reducidos de tiempos entre paquetes es necesaria una fuente altamente especializada. La construcción de los paquetes, por mínima que fuera (modificando un solo campo por iteración), el envío de mensajes a la salida estándar de la aplicación y la propia medida de los instantes de envío, con la correspondiente espera del tiempo teóricamente restante para procurar asegurar la correcta tasa de envío de los paquetes, provocaba por sí mismo una desviación enorme en los tiempos de envío.

Por eso ha sido necesario construir una fuente que no asegura calidad de servicio. Los paquetes se preparan antes de ser enviados, incluido el *timestamp*. No se realiza ningún tipo de monitorización del envío, para evitar las llamadas a dispositivos de entrada o salida innecesarios. Y no se utiliza ninguna llamada de espera, sino que se consulta continuamente el instante actual para comparar con el instante de envío teórico y así enviar el mensaje en el instante más preciso posible.

La situación en el lado de la recepción es similar. Las llamadas a dispositivos de entrada salida y el tratamiento de los datos hacían que los paquetes se acumularan a la entrada del receptor y los resultados salieran totalmente adulterados. La solución de nuevo consiste en crear un receptor no monitorizado y en almacenar los datos del paquete y el instante de recepción en memoria nada más recibirlos para pasar rápidamente a recibir el siguiente paquete, evitando el procesado hasta haber acabado de recibir todos los paquetes.

2.3.2. Aplicación *java*

La aplicación en *java* sobre la que se ha implementado la técnica de estimación *Pathload* sigue el paradigma cliente-servidor.

El cliente necesita tres datos para funcionar: La dirección de la máquina que contiene la parte servidor, el puerto TCP en el que la máquina servidor está esperando nuevas conexiones, y el puerto UDP local que será utilizado para capturar los trenes de paquetes.

La IP y el puerto de la máquina servidor son necesarios para poder entrar en contacto con ella. El puerto UDP local deberá quedar a la escucha, así que será necesario abrir ese puerto en los elementos con políticas de descarte de paquetes que se puedan tener (*firewall* de la máquina o la herramientas como *IPTables*).

El cliente también permite la introducción de parámetros para ajustar la estimación: Tasa inicial de testeo, número de paquetes por tren y número de trenes por prueba. De esta manera puede ajustar la precisión de la estimación, teniendo siempre en cuenta que un mayor número de trenes o pruebas significará casi siempre un tiempo mayor de estimación.

El servidor únicamente requiere de un puerto TCP donde situarse a la escucha de peticiones de estimación (Fig. 2.5).

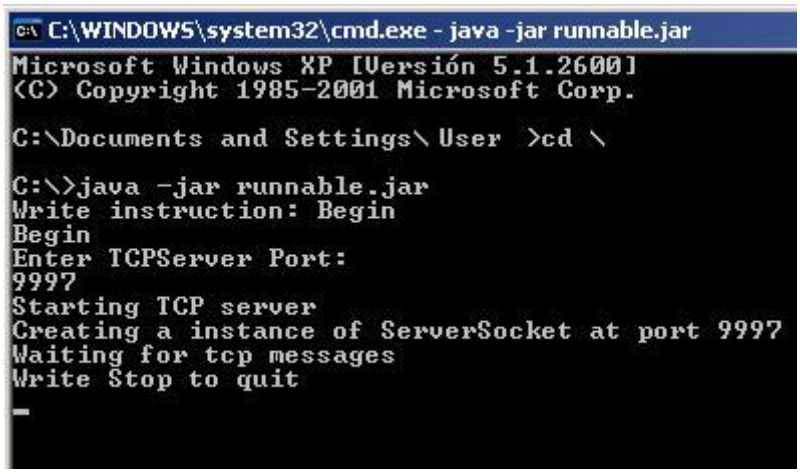

Fig. 2.5 Captura de la inicialización del servidor de estimación

2.3.3. Entorno de pruebas

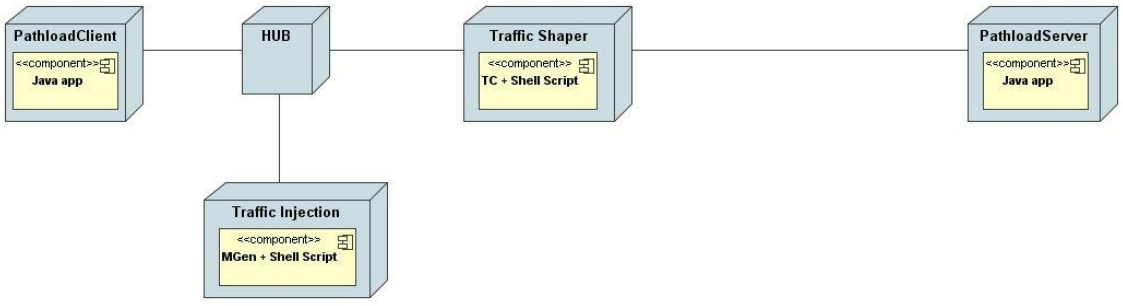

Fig. 2.6 Infraestructura para configurar los parámetros de la red de prueba y las características del tráfico cruzado

Puesto que las configuraciones por defecto de la mayoría de los dispositivos de red son limitadas y que el tráfico cruzado de una red es difícil de medir i

controlar, es necesaria una solución que permita tener todas estas variables bajo control. El objetivo de este escenario (Fig. 2.6) es proporcionar un entorno de pruebas controlado sobre el que probar la aplicación de estimación de ancho de banda.

Los componentes de este escenario son: Las partes cliente y servidor de la aplicación de estimación de ancho de banda; una máquina dedicada a la inyección controlada de tráfico cruzado en el escenario; y un conformador de tráfico que limitará las características originales de la red para adaptarlas a nuestras pruebas.

En el caso concreto del escenario que se ha utilizado, el conformador de tráfico se ha implementado sobre una máquina con dos interfaces de red, utilizando la aplicación TC para sistemas *Unix* y una serie de *scripts* para *Shell*. A través de estos dos elementos es posible limitar el ancho de banda, añadir retraso y *jitter*, generar pérdidas y definir los parámetros del *buffer* para los paquetes afectados.

El inyector de tráfico también se compone de *scripts* para *Shell*, aunque en este caso utiliza el programa *MGen* para la generación del tráfico cruzado. Con esto se pueden dar con facilidad instrucciones para la generación de tráfico periódico, tráfico de *poisson*, tráfico a ráfagas periódicas y tráfico aleatorio. Se puede determinar la tasa de transferencia a través del número de paquetes y del tamaño de los mismos.

El cliente y el servidor pertenecen a los componentes de la implementación de la aplicación para estimación de ancho de banda *Pathload*.

Las pruebas realizadas con tráfico constante de diferentes tasas en un canal de 50 Mbps son muy satisfactorias, dando un alto índice de relación entre el valor configurado y la estimación realizada (Fig. 2.8). La precisión parece ser mayor para el margen superior que para el inferior. Este último hecho es poco deseado, ya que el margen que procura una tasa de envío sin pérdidas es el inferior.

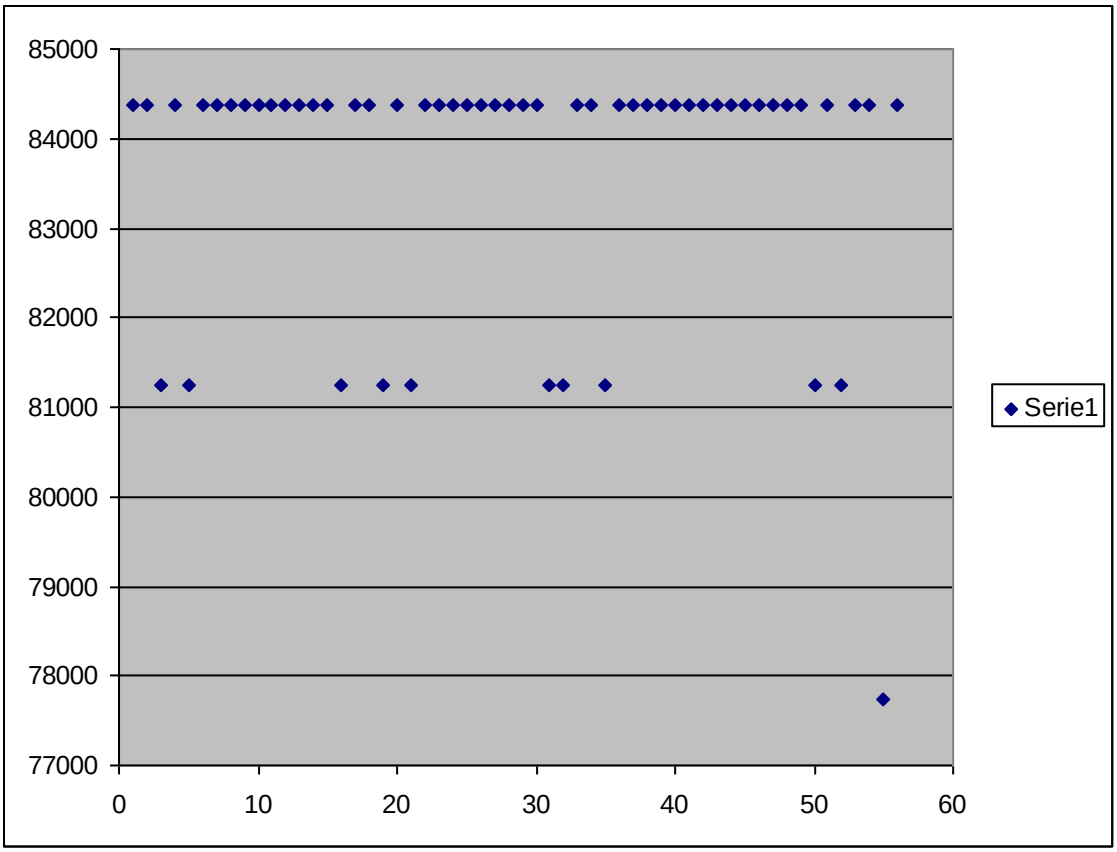

Fig. 2.7 Estimación del margen inferior para red limitada a 100 Mbps

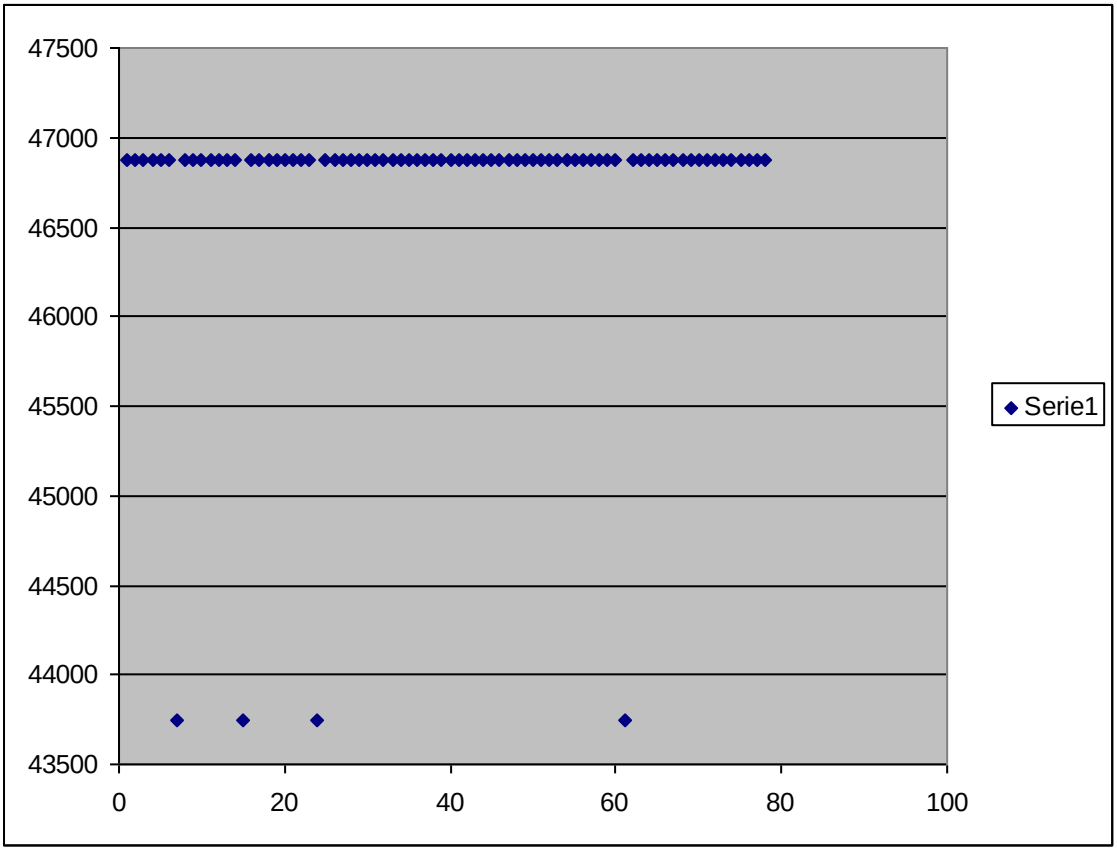

Fig. 2.8 Estimación del margen inferior para red limitada a 50 Mbps

Para comprobar la integridad de la aplicación durante largos periodos de ejecución, se ha utilizado un cliente modificado para que se ejecute de nuevo cada vez que terminaba un proceso de estimación. Este proceso se ha llevado a cabo sin ningún tipo de problema.

2.4. Servicios Web.

Los Servicios Web o WS sirven para realizar peticiones a objetos remotos. Utilizan el puerto 80 y estructura XML. Los WS indican los servicios que pueden prestar a través de un fichero WSDL, que sirve como interfaz para describir los métodos que tienen, los parámetros de entrada que requieren y el tipo de respuesta que van a devolver. Esto permite que la implementación del WS se pueda realizar en cualquier lenguaje de programación.

El medio seleccionado para hacer peticiones de contenido adaptado son los Servicios Web. También han sido escogidos para servir de interfaz de consulta de las estimaciones de ancho de banda.

Para implementar un Servicio Web se ha utilizado *Eclipse*, un entorno de desarrollo para *java*; el paquete de desarrollo de Servicios Web *JAX-WS*; y *Apache Tomcat*, un servidor JEE.

2.4.1. Cómo generar y desplegar un Servicio Web con *JAX-WS*

Con *Eclipse* se crea un *Dinamic web project* donde se construirá una clase *Java* que describa la implementación del servicio que prestará el Servicio Web [11]. Se deberá utilizar la anotación @WebService para que las aplicaciones del paquete de desarrollo *JAX-WS* puedan saber qué clase implementará el servicio, y habrá que copiar las librerías de *JAX-WS* a la carpeta “/WEB-INF/lib” del proyecto si no se encuentran en las librerías del servidor.

Utilizando la aplicación *wsgen* del paquete de desarrollo *JAX-WS* y esta clase, se generan el fichero WSDL, su XSD y las clases que realmente implementaran los métodos del Servicio Web. Posteriormente, se deberá modificar el fichero web.xml y crear el fichero sun-jax.xml para indicar la ruta de acceso al servicio, las clases que atenderán las peticiones y la clase donde se ha definido la implementación.

Para terminar de generar el Servicio Web, tan solo hace falta exportar el proyecto a un fichero “.war” con Eclipse.

Apache Tomcat se utiliza únicamente para desplegar los servicios. Los Servicios Web creados con el paquete de desarrollo *JAX-WS* pueden ser

desplegados simplemente moviendo el fichero ".war" generado por *Eclipse* a la carpeta webapps del directorio del *Tomcat*.

Para generar un cliente en *java* para este servicio, utilizar la aplicación *wsimport* del paquete de desarrollo de *JAX-WS* indicando la ruta al WSDL del servicio desplegado.

2.4.2. Servicios Web no bloqueantes: WS asíncronos

En WS las peticiones son bloqueantes, de manera que el hilo de ejecución queda bloqueado a la espera de una respuesta o de que pase un tiempo predeterminado (*timeout*). Esto significa que para peticiones que requieran procesos muy largos (consultas a bases de datos, procesos de transcodificación de media, etc) los procesos que las realizan no pueden hacer nada durante un extenso periodo de tiempo.

Los WS asíncronos [12] permiten al cliente invocar un servicio y continuar trabajando durante el tiempo que dura la petición. Al realizar la invocación, el cliente informa de la existencia de una instancia (*handler*) que se encargará de recoger la información de la respuesta del WS (Fig. 2.9). La implementación del *handler* definirá el tratamiento que se hará de los datos.

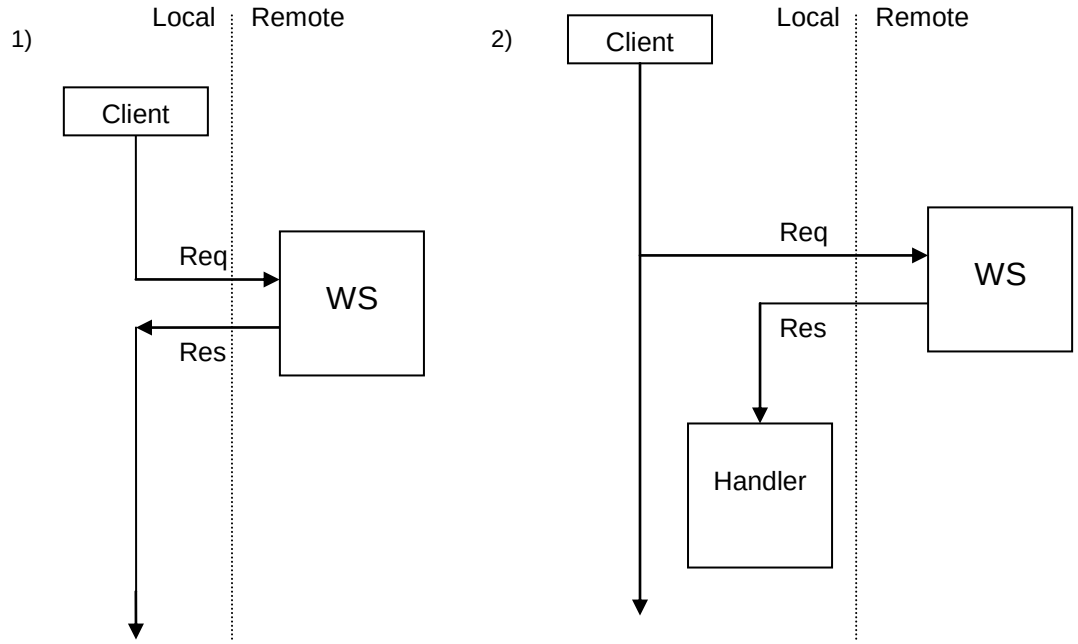

Fig. 2.9 Hilo de ejecución para cliente webservice síncrono 1) y cliente webservice asíncrono 2).

La herramienta *wsimport* del paquete de desarrollo para Servicios Web *JAX-WS* genera clientes con métodos de invocación asíncrona si el WSDL lo indica con el elemento:

```
<jaxws:bindings>
      <jaxws:enableAsyncMapping>true</jaxws:enableAsyncMapping>
```

```
</jaxws:bindings>
```

Este elemento se puede situar dentro de un método concreto para que solo afecte a ese método; o dentro del elemento *definitions*, de manera que todos los métodos descritos en el WSDL podrán ser invocados de forma asíncrona.

Esta configuración permite realizar diversas peticiones WS de larga duración para que se ejecuten en paralelo. De esta manera, se puede realizar una petición al controlador del estimador de ancho de banda para que informe del ancho de banda disponible para un cliente a la vez que se realiza una consulta a una base de datos para localizar un fichero de media seleccionado por el cliente. Finalmente, se puede ir preparando una petición al módulo de adaptación en la que se incluirán esos datos una vez se disponga de ellos.

2.4.3. Servicios Web sin contenedor JEE

Los WS suelen desplegarse sobre servidores JEE (Fig. 2.10). JEE es una plataforma de programación en *java* para servidores. Los servidores JEE tienen como objetivo proporcionar una infraestructura de acceso y servicios de gestión a los elementos que contienen, de manera que estos puedan centrarse en la lógica de negocio.

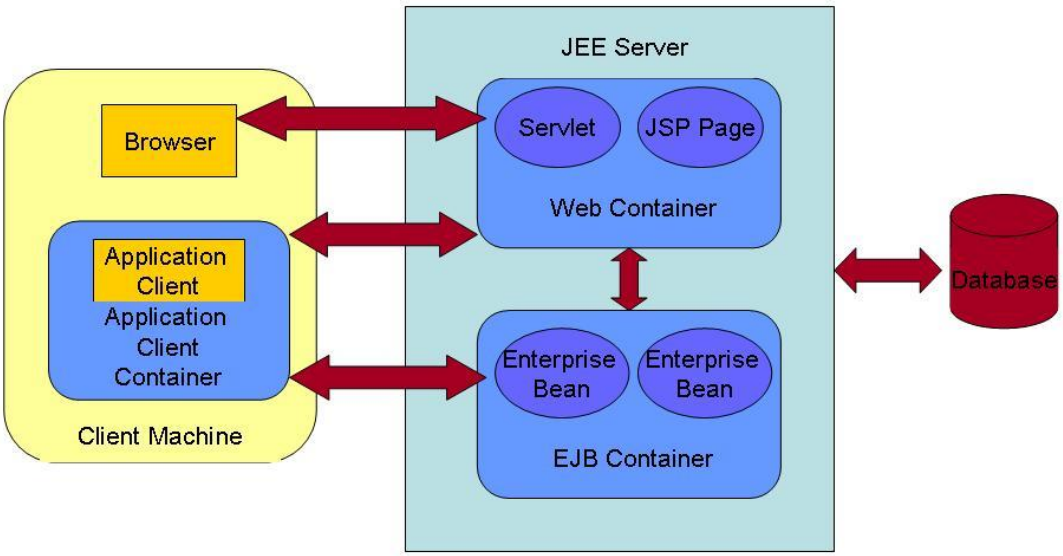

Fig. 2.10 Estructura servidor J2EE

Pero los WS también pueden funcionar sin instalar uno de estos servidores en la máquina donde van a estar [13]. Para ello se puede utilizar la clase

javax.xml.ws.Endpoint de la librería *JAX-WS*. Un WS construido con esta implementación se publica de forma automática sobre un servidor HTTP extremadamente ligero ya integrado.

Estos sistemas son ideales para entornos de características reducidas o aplicaciones que tengan como uno de sus objetivos ser lo más ligeras posibles.

El sistema se ha ideado con el objetivo de realizar tareas de monitorización del dispositivo que albergue el servicio. Se realizan peticiones al Servicio Web que informa del estado de las aplicaciones a las que está asociado. También se invocan métodos de inicialización de procesos.

Por tanto, es posible establecer un WS en el cliente de la aplicación de estimación que sea invocado por un servicio externo que procese los datos del estado. Esto podría servir para que una aplicación pudiera ir mostrando el estado de los usuarios activos y lo fuera actualizando periódicamente. Además, al utilizar el puerto 80 se evitarían muchas de las medidas de seguridad que suelen bloquear aplicaciones que se conecten a puertos no estandarizados o bien conocidos.

2.4.4. Módulo de adaptación

El módulo de adaptación ofrece dos servicios: “Listar contenidos” y “Obtener contenido”.

Listar contenidos devuelve un listado con los contenidos de los que se tenga constancia en la base de datos.

Obtener contenido inicia el proceso de adaptación. La respuesta a este proceso consiste en una URL de tipo RTSP que indica el lugar al que la aplicación de reproducción de *streaming* deberá conectarse.

2.4.5. Módulo de estimación

El módulo de estimación ofrece dos servicios: “Pedir estimación” o “Pedir a base de datos”.

Pedir estimación espera a que se realice una estimación a un cliente indicado y obtiene los resultados de esa estimación.

Pedir a base de datos calcula una media de los resultados de estimaciones anteriores para un cliente si tienen cierta correlación y si no están relacionados devuelve el último resultado.

2.5. Servicio de contenidos

El servicio de contenidos ofrece una lista de contenidos y proporciona un acceso en *streaming* a ese contenido, adaptado a la tasa que mejor se corresponde con la red del usuario. El servicio de contenidos está formado por

dos módulos, el de adaptación y el de transcodificación, y utiliza los servicios del módulo de estimación.

El módulo de adaptación proporciona servicios de consulta de posibles contenidos y de acceso a contenido específico.

Para la consulta de contenidos, se conecta a una base de datos y obtiene el nombre, el tipo y la localización de los contenidos existentes.

Para el acceso a un contenido específico, inicia una petición de estimación mientras que se conecta a la base de datos para obtener la localización y el tipo de media. Con esa información, realiza una petición de transcodificación al módulo de transcodificación. La respuesta a la petición es una ruta para la reproducción de la media adaptada en *streaming*. Esa misma respuesta es enviada hacia el cliente como ruta de acceso al contenido.

2.5.1. Servidor de transcodificación *Alembik*

Alembik es un servidor de transcodificación. Así, su función principal es recibir peticiones que contendrán uno o varios trabajos de transcodificación, comunicarse con aplicaciones de transcodificación para que los realicen, informar en caso de consulta del estado de un trabajo en curso y responder con una URL apuntando al trabajo realizado o un código de error indicando por que ha fallado la transcodificación.

2.5.1.1 Software asociado

Como ya se ha dicho, Alembik utiliza aplicaciones externas para realizar las tareas de transcodifcación. Según las necesidades del sistema será necesario incorporar software de transcodificación de video, de audio i/o de imagen.

El software utilizado para realizar las transcodificaciones de imagen es *ImageMagick*. *ImageMagick* es una herramienta de ejecución por consola que permite crear, modificar y dar formato a imágenes, con soporte para una gran cantidad de formatos.

El software utilizado para realizar las transcodificaciones de audio y video es *Ffmpeg*, una aplicación que permite realizar transformaciones sobre audio y video.

También es necesario instalar ciertas aplicaciones externas si se desea que la máquina en que se encuentra *Alembik* pueda servir los contenidos transcodificados en *streaming*. Estas aplicaciones son *MP4Box*, que genera flujos de media en formato mpeg-4; y *Darwin Streaming Server*, que atiende las peticiones de video en *streaming* y soporta el protocolo RTSP para el envío y el control de flujos de media.

2.5.1.2 API para peticiones de transcodificación

Las peticiones de transcodificación están basadas en la especificación OMA STI. Para saber que valores deben ser usados en cada atributo, consultar la documentación relacionada [25].

Se pueden realizar peticiones por tres vías:

Se puede implementar un cliente *java* que utilice la librería descargada con el código de *Alembik* "alembik-soap-client.jar". Se utiliza la clase *TranscodingManager* para enviar peticiones representadas en objetos de la clase *TranscodingRequest*. Los resultados son devueltos en objetos de tipo *TranscodingResponse*. Las peticiones pueden ser síncronas o asíncronas. También se puede consultar el estado de las peticiones ya enviadas para ver si ya se ha obtenido el fichero de origen y si ya se ha realizado la transcodificación.

También se puede acceder a los servicios del *framework* a través de Servicios Web. Para ello, se debe utilizar la descripción que se encuentra en:

http://[host]:[port]/alembik/process?wsdl

Para poder generar un cliente de ese Servicio Web.

Finalmente, se pueden realizar peticiones de transcodificación a través de peticiones HTTP a una serie de *Servlets* que despliega el *framework*. Los *Servlets* disponibles son:

http://[host]:[port]/alembik/transcode: *Servlet* que procesa peticiones síncronas. Devuelve una ruta al resultado de la transcodificación.

http://[host]:[port]/alembik/asyncTranscode: *Servlet* que procesa peticiones asíncronas. No funciona en *Apache Tomcat*.

http://[host]:[port]/alembik/isReady: *Servlet* que responde a la consulta de estado para una petición. Puede devolver la ruta al resultado de la transcodificación si ésta ha terminado; una página de estado que indica el estado de la petición en el momento de realizar la consulta (porcentaje y fase); o una página de error si el proceso de transcodificación no se puede llevar a cabo.

Capítulo 3. PLANIFICACIÓN Y COSTES

En este capítulo se muestra una visión más general de las tareas llevadas a cabo durante la realización del proyecto. Para facilitar su representación estas tareas se engloban en los siguientes tipos:

Estudio previo: Tareas durante las cuales la mayor parte del tiempo se dedica la búsqueda de tecnologías e información que puedan resultar útiles para los objetivos del proyecto.

Diseño: Tareas cuyo principal objetivo es plasmar la información obtenida en documentos o esquemas que servirán de guía a la hora de implementar los componentes del sistema. También se realizan pequeñas implementaciones de escenarios de prueba para comprobar que su funcionamiento se corresponde con el que describe la información recogida durante el estudio previo.

Investigación: Tareas de búsqueda de información surgidas por una variación en la planificación del proyecto.

Implementación: Producción del software ideado durante la etapa de diseño. Directamente ligado a la etapa de testeo debido al sistema de desarrollo en módulos del sistema.

Testeo: Tareas de realización de pruebas para los diversos módulos implementados.

Documentación: Redacción de esta memoria y representación en esquemas de los componentes finales implementados.

3.1 Dedicación

La siguiente figura (Fig. 3.1) es una representación de la distribución que se ha hecho del tiempo que se ha dedicado al proyecto.

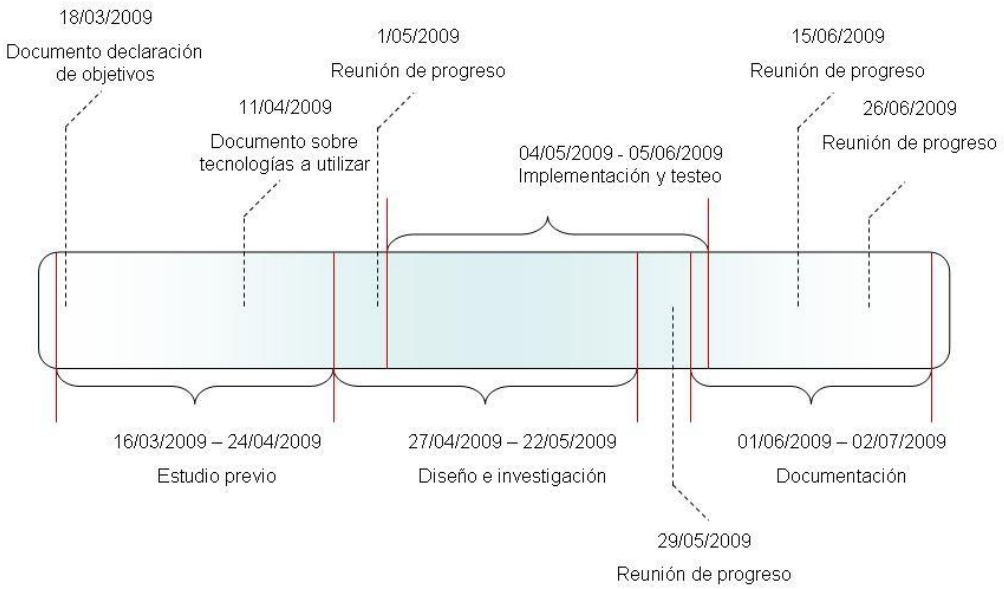

Fig. 3.1 Distribución temporal de los diversos bloques de tareas

A pesar de que se realizó una primera planificación de las tareas a realizar, finalmente la distribución se ha basado más en los resultados las reuniones de progreso y la evaluación de los documentos presentados.

Durante la etapa de estudio previo se fueron concretando objetivos y tecnologías que pudieran ser relevantes para el proyecto.

Tras acotar los caminos que se podían seguir, se inicio una etapa de diseño. Una vez que se empezó a tener definiciones claras de algunos componentes, se comenzó a alternar el diseño de nuevos componentes con la implementación, el testeo y la revisión de los componentes ya diseñados.

Finalizado el diseño de nuevos componentes, se prosiguió con la implementación y el testeo de los ya existentes, mientras que se iniciaba la redacción de los diversos capítulos de esta memoria.

3.2 Tareas y distribución temporal

Tabla 3.1. Desglose de tareas realizadas y tiempo que se les ha dedicado.

Tipo de tarea	Descripción	Tiempo
Estudio previo	Discusión y definición de los objetivos del proyecto.	10 h

Estudio previo	Primera búsqueda de información y tecnologías útiles.	40 h
Estudio previo	Redacción de documento comparativo de tecnologías útiles.	8 h
Estudio previo	Discusión sobre documento comparativo para eliminación y propuesta de tecnologías útiles.	4 h
Estudio previo	Búsqueda final de tecnologías útiles propuestas e información sobre ellas.	60 h
Diseño	Diseño de escenarios de captura de paquetes.	10 h
Diseño	Diseño de escenario de gestión remota.	10 h
Implementación y testeo	Construcción de capturador de paquetes y primeras pruebas.	12 h
Diseño e implementación	Diseño e implementación de Servicio Web asíncrono para evaluar su integración.	16 h
Implementación y testeo	Construcción de gestor remoto y pruebas junto a capturador de paquetes.	12 h
Diseño e implementación	Diseño e implementación de Servicio Web sin contenedor JEE para evaluar su integración.	16 h
Investigación	Capturador de paquetes resulta no ser válido para objetivos del proyecto. Búsqueda de nuevos métodos de estimación de QOS.	12 h
Diseño	Estudio y diseño de aplicación de estimación de ancho de banda por técnica *Packet Pair.*	16 h
Implementación y testeo	Implementación y testeo de aplicación por *Packet Pair*.	16 h
Diseño	Estudio y diseño de aplicación de estimación de ancho de banda por técnica *Pathload*.	20 h
Implementación y testeo	Implementación y testeo inicial de aplicación por *Pathload*. Selección de técnica para estimación.	12 h
Investigación	Búsqueda de tecnologías para la evaluación de técnica de estimación de ancho de banda.	4 h
Diseño	Diseño de escenario de pruebas para aplicación de estimación de ancho de banda.	4 h
Implementación y testeo	Montaje de escenario de pruebas. Prueba de los componentes para asegurar que los resultados de la evaluación de las herramientas desarrolladas serán válidos.	8 h
Implementación y testeo	Pruebas y optimización de aplicación de estimación.	20 h
Investigación	Selección y descarte de tecnologías para transcodificación multimedia.	2 h
Investigación	Investigación de *framework* para gestión de peticiones de transcodificación *Alembik* 1.0 beta 4.	12 h
Implementación	Primer despliegue de *framework Alembik*. Interrumpida por problemas con la máquina de desarrollo.	6 h
Implementación y testeo	Segundo despliegue de *framework Alembik*. Aprendizaje del proceso de instalación y de la	16 h

	API de consulta.	
Implementación, testeo y documentación	Despliegue final de *framework Alembik* con soporte para peticiones de transcodificación multimedia y retransmisión en *streaming*. Redacción de guía de instalación.	12 h
Diseño	Diseño de posible implementación para módulo de unión entre estimador de ancho de banda y *framework* de gestión de peticiones de transcodificación.	8 h
Documentación	Redacción de la memoria.	110 h
	Total	476 h

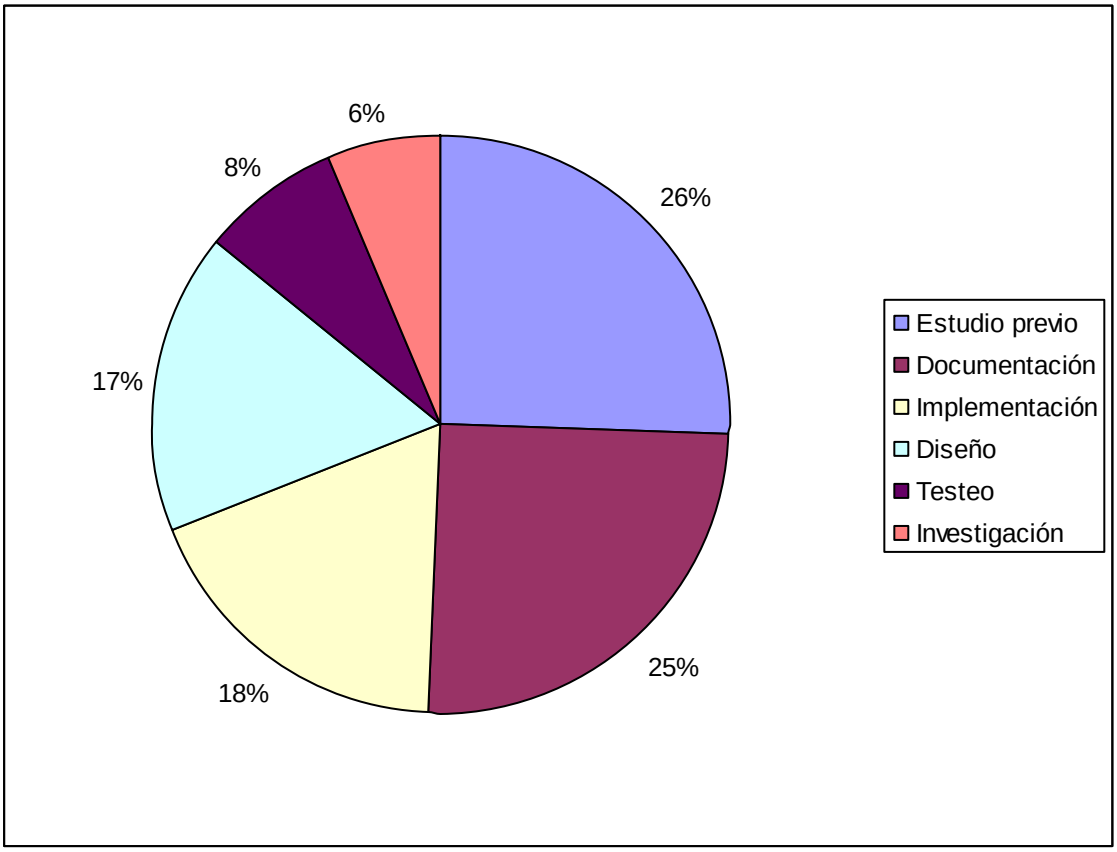

Fig. 3.2 Porcentaje de dedicación por grupos de tareas

La dedicación total del proyecto supera los requerimientos mínimos de carga lectiva para un TFC de Ingeniería Técnica de Telecomunicaciones, que es de 420 horas.

Como se puede observar (Fig. 3.2) las tareas de estudio previo han ocupado una gran parte del desarrollo del proyecto, dado que ha habido una gran carga de investigación de tecnologías existentes y búsqueda de soluciones conocidas para determinar la calidad de servicio.

La carga de diseño es casi tan grande como la de implementación porque ha sido necesario aprender desde cero muchas de las tecnologías utilizadas,

obligando a repetir partes ya diseñadas a medida que se iban dominando los nuevos conocimientos.

Finalmente, han sido necesarias ciertas tareas de investigación durante el proyecto tanto por la inclusión de nuevos objetivos (como el de añadir el *framework Alembik* al sistema) como por la necesidad de buscar tecnologías substitutivas para reemplazar las que no han acabado de funcionar como se tenia previsto.

3.3 Estimación de costes

Para el análisis de costes se tendrán en cuenta los siguientes factores:

- **Tiempo dedicado**: El tiempo total dedicado desde el día de inicio del proyecto hasta el final de la redacción de esta memoria es de 75 días, con jornadas de trabajo de entre 4 y 8 horas diarias, dado que se ha compaginado la realización del TFC con una experiencia laboral.

  Esto ha limitado la implementación de todos los elementos del sistema, de manera que se ha centrado el trabajo en la construcción de los elementos diferenciadores (estimador de ancho de banda y gestor de peticiones de transcodificación) y únicamente se han diseñado y buscado tecnologías que podrían resultar útiles para los elementos que los conectarían.

- **Material utilizado**: Como se ha comentado en el apartado 2.1, se ha utilizado dos máquinas fijas proporcionadas por el espacio de trabajo de la fundación i2CAT en la EPSC, además de algunas otras máquinas en ocasiones puntuales.

  Todas estas máquinas son compartidas entre varios proyectos, se han utilizado con anterioridad y se seguirán utilizando en un futuro, de manera que el impacto de su utilización es mínimo y se incluye dentro del coste de compra de la máquina que asumió i2CAT en el momento de su adquisición.

  La conexión a la red también se ha realizado desde el espacio de trabajo de i2CAT en la EPSC, pero se trata de un recurso compartido para todas las máquinas del laboratorio y es muy difícil determinar los costes relacionados. La misma situación se produce con los costes de electricidad, servicios para el trabajador, etc.

  El software utilizado durante la realización del proyecto esta registrado bajo licencias de software libre (como el paquete de desarrollo de Servicios Web *JAX-WS*), utiliza licencias para estudiantes (como es el caso del software de ofimática *Microsoft Office* 2003/2007) o se trata de herramientas gratuitas (que seria el caso del IDE *Eclipse*).

  Por tanto, no se pueden adjudicar ningún coste añadido al proyecto por recursos utilizados.

- **Personal**: El precio por hora medio en el mercado para un estudiante de Ingeniería Técnica sin titulación es de unos 6 euros.

- **Desplazamientos**: El desplazamiento desde la vivienda hasta el espacio de trabajo se ha realizado en coche. Los costes del carburante son de unos 10 a 12 euros a la semana, sin contar con los costes de seguro y mantenimiento del coche.

  Por otra parte, el espacio de trabajo es a la vez el lugar donde se realiza la actividad laboral diaria, de manera que el desplazamiento se tendría que realizar igualmente. Por tanto, estos costes tampoco pueden ser ligados expresamente al desarrollo del proyecto.

Tras este desglose, se observa que los únicos costes que se pueden asociar al proyecto son los costes de personal por tiempo trabajado. Dado que solo un estudiante ha realizado todo el trabajo, el coste final queda de la siguiente manera:

- Horas diarias dedicadas: 6 horas
- Días totales dedicados al proyecto: 75 días
- Precio por hora para un estudiante de ingeniería técnica: 6 € / hora

- TOTAL: 2700 €

Capítulo 4. CONCLUSIONES

4.1 Objetivos cumplidos

El objetivo de este TFC era el de diseñar una infraestructura viable para la gestión de la calidad de servicio que debía ofrecerse a cada usuario de un servicio multimedia.

El servicio multimedia seleccionado ha sido el video bajo demanda, el factor de decisión han sido los parámetros de la red del usuario y la calidad de servicio ofrecida se ha relacionado directamente con la tasa de envío de los contenidos, de manera que a mejores parámetros de red, mayor tasa y mejor calidad para el usuario.

Se han diseñado, implementado y desplegado los elementos que realmente diferencian este sistema de los demás, que son el estimador de ancho de banda y el servidor de transcodificación. También se han hecho pruebas de tecnologías que pudieran servir para implementar el módulo de enlace entre estos elementos.

Por falta de tiempo, la implementación del sistema completo y otras mejoras quedan como trabajo futuro.

4.2 Trabajo futuro

- Implementar el módulo de enlace entre el estimador de ancho de banda, el repositorio de contenidos y el servidor de transcodificación.

- Completar el escenario con un sistema de identificador de usuarios.

- Añadir elementos de persistencia al módulo de estimación (como puede ser una base de datos) para poder realizar análisis estadísticos del ancho de banda disponible para usuarios concretos, grupos de usuarios, determinadas redes, etc.

- Seguir estudiando nuevas técnicas de estimación del ancho de banda que carezcan de las limitaciones de las que se han probado en este proyecto.

- Añadir elementos de monitorización del sistema.

4.3 Impacto medioambiental

El sistema de gestión de la calidad de servicio presentado es además un sistema de optimización de recursos. Controlando las tasas a las que se sirve cada contenido para que no produzcan errores en la comunicación o en el

tráfico adyacente, no solo se produce un ahorro energético por el menor consumo de los servidores de media en el envío de contenidos a altas tasas de transferencia para dispositivos que no las aprovechaban. También se producen un el menor número de retransmisiones por las pérdidas que se podrían generar en caso de no utilizarse.

Y lo más importante de todo: al utilizar de forma más eficiente las redes de comunicación existentes, permite que estas se puedan seguir utilizando durante un tiempo mayor. De esta manera, se minimiza el impacto que tiene la renovación de las infraestructuras de comunicación

Así, se logran dos objetivos. Por un lado, se evita o como mínimo se pospone el enorme consumo energético y la gran cantidad de contaminación - principalmente del aire, por la emisión de gases provenientes del consumo de combustibles fósiles y el envío de grandes cantidades de polvo a la atmósfera - que generaría una obra del tamaño de las infraestructuras de comunicaciones existentes, aunque solo se tratara de las porciones más antiguas.

Y por la otra, se conseguiría tiempo para buscar una solución mejor a los sistemas de comunicación actuales, ya sea en forma de medios de transmisión más eficientes, menos contaminantes en su proceso de fabricación, más fáciles de integrar en el medio ambiente, etc.

4.4 Conclusiones personales

Este TFC es el primer gran proyecto de trabajo individual que he realizado en toda la carrera. Eso lo ha convertido en un trabajo muy enriquecedor, puesto que la ausencia de división de tareas me ha otorgado un punto de vista más general de la gestión de proyectos.

También hay que destacar que aunque este proyecto se haya llevado a cabo de forma individual, siempre se ha contado con el apoyo y la ayuda tanto del tutor como de los compañeros de i2CAT. Gracias a ellos la motivación por completar el proyecto ha sido mayor y sus conocimientos han servido tanto para evaluar otros puntos de vista como para formarme a mí mismo.

La proporción de objetivos cumplidos me ha parecido adecuada para un proyecto de estas características, pero debo admitir que me hubiera gustado poder seguir trabajando en él algo más de tiempo. Con esto he aprendido otra lección. Y es que hay que saber cuándo terminar un proyecto, marcarse objetivos a corto/medio plazo y centrarse en cumplirlos. Porque si solo se tiene como objetivo la implementación del sistema completo, el proyecto no terminará nunca. Siempre quedará algo por hacer, y hay que aceptarlo.

Así mismo, se debe destacar que el trabajo se ha realizado a la par con una actividad laboral que, si bien puede que haya limitado en algún momento la dedicación que se podía prestar al proyecto, indudablemente me ha enriquecido tanto en conocimientos como en experiencias y me ha dado

herramientas para gestionar situaciones con las que indudablemente me voy a encontrar en un futuro.

Por tanto mi valoración general es positiva y agradezco haber tenido la oportunidad de vivir todas estas situaciones.

REFERENCIAS

Artículos consultados

[1] Wenyu Jiang, Henning *Schulzrinne, "QoS Measurement of Internet Real.Time Mulimedia Services*", 1999

[2] Sing Li, "*From black boxes to enterprises*", 2002

[3] Manish Jain, Constantinos Dovrolis, *"Pathload: a measurement tool for end-to-end available bandwith"*, 2002

[4] Kevin I-Sen Lai, "*Measuring The Bandwidth Of Packet Switched Networks*", 2002

[5] R Prasad, C Dovrolis, M Murray, K Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools", IEEE network, 2003

[6] Alok Shriram, Margaret Murray, Young Hyun, Nevil Brownlee, Andre Broido, Marina Fomenkov, Kc Claffy *"Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links"*, 2005

[7] André Ríos, Jesús Alcober, Antoni Oller, Victor Sempere, "*Design and Implementation of a measurement and alert system of QoS parameters in SIP based Internet Telephony*", 2006

[8] Spirent Communications Inc, "*Measuring Jitter Accurately*", 2007

[9] Alberto Toro Sánchez, "*Arquitectura avanzada de adaptación de recursos multimedia*" 2008.

[10] E. M. Macías, A. Suárez, "*Estimación proactiva de la calidad de recepción de video streaming en WiFi*"

Tutoriales

[11] Glassfish project, JAX-WS "*User guide*", Último acceso 02/07/09

[12] IBM WebSphere Application Server, Feature Pack for Web Services, "*Desarrollo y despliegue de aplicaciónes: Servicios Web*", Último acceso 02/07/09

[13] java.net, "*JAX-WS Web Services Without Java EE Containers*", Último acceso 02/07/09

[14] Sun Microsystems, Inc, "*Java Management Extensions (JMX) Technology Tutorial*", Último acceso 02/07/09

[15] Jpcap, "*Jpcap Tutorial*", Último acceso 02/07/09

[16] MythTV project, "Setup *Darwin Streaming Server and MP4Box*", Último acceso 02/07/09

[17] Rafael Sanches, "*Install ffmpeg for alembik transcoding server with 51.40.4 on ubuntu*", Último acceso 02/07/09

Enlaces Web

[18] GlassFish Project, HTTP/S JMX Connector home page, URL: <https://glassfish.dev.java.net/javaee5/admin-infra/http-jmx-connector.html>, Último acceso 02/07/09

[19] java.net, ws-jmx-connector, URL: <https://ws-jmx-connector.dev.java.net>, Último acceso 02/07/09

[20] MX4J, Open Source JMX for Enterprise Computing, URL: <http://mx4j.sourceforge.net/>, Último acceso 02/07/09

[21] The Wireshark Wiki, RTP statistics, URL: <http://wiki.wireshark.org/RTP_statistics>, Último acceso 02/07/09

[22] Wireshark source code repository, URL: <http://anonsvn.wireshark.org/viewvc>, Último acceso 02/07/09

[23] Jpcap home page, URL: <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html>, Último acceso 02/07/09

[24] jNetPcap home page, URL: <http://jnetpcap.com/>, Último acceso 02/07/09

[25] Open Mobile Aliance home, URL: <http://www.openmobilealliance.org/>, Último acceso 02/07/09

ANEXO

Anexo 1: ACRÓNIMOS

QoS: Quality of Service

IP: Internet Protocol

API: Application Programing Interface

GPS: Global Position System

PGM: Packet Gap Method

PRM; Packet Rate Method

UDP: User Datagram Protocol

FIFO: First In First Out

SNMP: Simple Network Management Protocol

IBM: International Business Machines

HP: Hewlett-Packard

JMX: Java Management eXtensions

RMI: Remote Method Invocation

JSR: Java Specification Request

HTTP: HyperText Transfer Protocol

SOAP: Simple Object Access Protocol

WURFL: Wireless Universal Resource FiLe

XML: Extensible Markup Language

XSD: XML Schema Definition

TFC: Trabajo de Final de Carrera

EPSC: Escola Politècnica Superior de Castelldefels

VNC: Virtual Network Computing

SSH: Secure Shell

JEE: Java Enterprise Edition

WS: Web Service

JDK: Java Development Kit

JRE: Java Runtime Enviroment

TCP: Transmission Control Protocol

TC: Traffic Control

WSDL: Web Service Description Language

RTSP: Real Time Streaming Protocol

URL: Uniform Resource Locator

IDE: integrated development environment

OMA STI: Open Mobile Aliance Standard Transcoding Interface

Anexo 2: SCRIPTS PARA INYECCIÓN Y CONFORMACIÓN DE TRÁFICO

scriptarpbridge.sh

Convierte una máquina con dos interfaces de red en un bridge a nivel lógico.

```
#! /bin/bash

ifconfig eth0 down
ifconfig eth1 down
ifconfig br0 down

echo "Setting ARP Bridge....."
echo "Arrancando interfaces"
ifconfig eth0 up 0.0.0.0
ifconfig eth1 up 0.0.0.0

echo "Creando interfaz virtual"
brctl addbr br0

echo "AÃ±adir interfaces al bridge"
brctl addif br0 eth0
brctl addif br0 eth1

echo "Arrancando la interfaz virtual"
ifconfig br0 up
```

stopbridgeARP.sh

Elimina el bridge a nivel lógico y pide IP para una de las interfaces de red.

```
#! /bin/bash

ifconfig eth0 down
ifconfig eth1 down
ifconfig br0 down
brctl delbr br0
echo "Bridge eliminado"
ifconfig eth0 up
ifconfig eth1 up
dhclient eth1
```

scriptmgenstart.sh

Script para generación de tráfico con múltiples formas.

```
#!/bin/bash


echo "Ejecutar desde el mismo directorio donde se encuentre mgen"

read -p "Introduce IP destino " dstIP
read -p "Introduce el puerto destino " dstPort
read -p "Â¿Cuantos paquetes por segundo? " numPackets
read -p "Â¿De que tamaÃ±o? " packetSize
read -p "Introduce 1 para distribucion periodica, 2 para distribuciond de
poisson, 3 para rafagas, 4 para aleatorio " dist

distValue="periodic"
burstType="regular"
burstInterval=1.0
burstDurationType="fixed"
burstDuration=1.0

case $dist in
1)distValue="periodic";;
2)distValue="poisson";;
3)read -p "Â¿Cada cuanto habra una rafaga? (en segundos) " burstInterval
read -p "Introduce la duracion en segundos de la rafaga " burstDuration;;
4)distValue="poisson"
burstType="random"
burstDurationType="exponential";;
*)echo "error en distribution";;
esac

./mgen instance mgen1 event "0.0 on 1 udp dst "$dstIP"/"$dstPort" burst
["$burstType" "$burstInterval"  "$distValue" ["$numPackets"
"$packetSize"] "$burstDurationType" "$burstDuration"]"
```

scriptmgenstop.sh

Detiene el proceso mgen en curso.

```
#!/bin/bash

./mgen instance mgen1 event "off 1"
```

scriptvelocidadesYperdidas.sh

Proporciona la interfaz de configuración para conformación de tráfico.

```
#!/bin/bash

tc qdisc del dev eth0 root
tc qdisc del dev eth1 root
tc qdisc del dev br0 root
read -p "Introduce la velocidad de eth0 en kbit " speedeth0
read -p "Introduce la velocidad de eth1 en kbit " speedeth1
read -p "Introduce si quieres tener delay en eth0 el valor en ms...y si no 0 "
delayeth0
read -p "Introduce si quieres tener delay en eth1 el valor en ms...y si no 0 "
delayeth1
read -p "Introduce si quieres tener jitter en eth0 el valor en us...y si no 0 "
jittereth0
read -p "Introduce si quieres tener jitter en eth1 el valor en us...y si no 0 "
jittereth1
read -p "Introduce el tamaÃ±o del buffer en eth0 en bytes...y si no 0 "
buffereth0
read -p "Introduce el tamaÃ±o del buffer en eth1 en bytes...y si no 0 "
buffereth1
read -p "Introduce el valor de limit en eth0...y si no 0 " limiteth0
read -p "Introduce el valor de limit en eth1...y si no 0 " limiteth1
read -p "Introduce el valor de perdidas en eth0 en %...y si no 0% "
perdidaseth0
read -p "Introduce el valor de perdidas en eth1 en %...y si no 0% "
perdidaseth1

echo "Se crearÃ¡n las entradas"
tc qdisc add dev eth0 root handle 1:0 netem delay "$delayeth0"ms
"$jittereth0"us loss $perdidaseth0
tc qdisc add dev eth0 parent 1:1 handle 10: tbf rate "$speedeth0"kbit buffer
$buffereth0 limit $limiteth0
tc qdisc add dev eth1 root handle 1:0 netem delay "$delayeth1"ms
"$jittereth1"us loss $perdidaseth1
tc qdisc add dev eth1 parent 1:1 handle 10: tbf rate "$speedeth1"kbit buffer
$buffereth1 limit $limiteth1
echo "Mostrando colas"
tc qdisc show
```

Anexo 3: PROCESO DE INSTALACIÓN

Proceso de instalación del *framework* de gestión de peticiones de transcodificación *Alembik* en una máquina Linux (Ubuntu 8.04).

Preparando el entorno

Instalar la máquina virtual de *Java*.

sudo apt-get install sun-java6-sdk

Definir la variable de entorno JRE_HOME (para máquina virtual solo) o JAVA_HOME (para SDK) para el directorio en el que se ha instalado. Dado que luego se utilizará el usuario root para poder arrancar el servidor, una forma de definir estas variables es añadiéndolas al fichero ".bashrc" del usuario root.

export JRE_HOME="ruta de jre"

export JAVA_HOME="ruta de sdk"

Instalar un servidor Apache tomcat 5.5.2x, Glassfish v2 o JBoss 4.2.x GA. En este ejemplo se utiliza tomcat 5.5.27. Para ello, se descarga el paquete Core de la página:

http://tomcat.apache.org/download-55.cgi

Para que el servidor pueda dar soporte para los Servicios Web utilizados en este *framework*, se necesitan una serie de librerías de la API de Servicios Web JAXWS 2.x. Descargar el binario y ejecutar para obtener un directorio.

Abrir y copiar todo el contenido de la carpeta "lib" a la carpeta del mismo nombre que se encuentra dentro del directorio del tomcat (para Ubuntu 8.04: '%tomcat_install_dir%/shared/lib/').

Aplicaciones de transcodificación

Atención: Buena parte de los pasos siguientes requieren permisos de súper usuario. Si no se es el usuario root, hay que acordarse de añadir "sudo" antes de ejecutar los comandos.

FFMPEG:

Como *Alembik* solo soporta las versiones de la librería libavcodec 51.40.4 y 51.48.0 para *ffmpeg*, y estas son versiones antiguas, se debe descargar el código fuente que contenga alguna de esas versiones y compilarlo [17]. También serán necesarios algunos elementos externos.

Descargar e instalar los paquetes de amr:

wget http://ftp.penguin.cz/pub/users/utx/amr/amrnb-6.1.0.4.tar.bz2

wget http://ftp.penguin.cz/pub/users/utx/amr/amrwb-7.0.0.1.tar.bz2

Para descomprimirlo puede usarse bzip2. Ejecutar el “configure” y después compilar a partir del make creado con “make” y “make install”

Descargar el resto de elementos externos necesarios para compilar *ffmpeg*:

apt-get install g++ nasm zlib1g-dev libx264-dev libfaad-dev faad liba52-0.7.4-dev libgsm1-dev libmp3lame-dev libtheora-dev libvorbis-dev libxvidcore4-dev libfaac-dev libimlib2-dev libfreetype6-dev

Atencion: libmp3lame-dev puede no aparecer con ese nombre en algunos repositorios. Probar con liblame-dev si aparece ese error.

Descargar el código fuente de *ffmpeg* en la versión adecuada:

wget http://dag.wieers.com/rpm/packages/ffmpeg/ffmpeg-0.4.9-070530.rf.src.rpm

Descomprimir (con algún programa como Alien para pasar de rpm a tar.gz) hasta obtener el directorio “ffmpeg-20070530”.

Configurar el *ffmpeg* antes de compilarlo (desde el directorio descomprimido de *ffmpeg*):

./configure --enable-libmp3lame --enable-libogg --enable-libvorbis --enable-libamr-nb --enable-libfaad --enable-libfaac --enable-libgsm --enable-xvid --enable-x264 --enable-liba52 --enable-liba52bin --enable-pp --enable-shared --enable-pthreads --enable-libtheora --enable-gpl --disable-strip --enable-libfaadbin --prefix=/usr/local --enable-swscaler --enable-libamr-wb

Atención: Tener en cuenta que se debe modificar este configure si se ha tenido el problema con la librería libmp3lame.

Instalar con “make” y “make install”.

ImageMagick:

Instalar por repositorio.

MP4Box y Darwin Stream server [16]:

Descargar el código fuente del proyecto en el que se integra *MP4Box*

http://gpac.sourceforge.net/home_download.php

Compilar e instalar. El ejecutable resultante se encuentra en:

$gpac_directory/bin/gcc/MP4Box

Descargar el código fuente de *Darwin Streaming Server* (6.0.3) para linux:

http://dss.macosforge.org/

Crear un grupo y un usuario para DSS:

$groupadd qtss

$useradd qtss -g qtss

Ir al directorio con el código fuente de DSS y descargar los parches que sean necesarios:

Para linux:

http://www.abrahamsson.com/dss-6.0.3.patch

patch -p1 < dss-6.0.3.patch

Para máquinas de 64 bits:

http://www.abrahamsson.com/dss-6.0.3-x86_64.patch

patch -p1 < dss-6.0.3-x86_64.patch

Compilar el código:

./buildtarball

Ir al directorio DarwinStreamingSrvr-Linux y ejecutar el script Install.

Se recomienda realizar lo que queda de proceso una vez que se haya desplegado el servicio *Alembik*.

Conectarse a http://localhost:1220 y completar la instalación seleccionando la carpeta “storage” del directorio alembik como directorio para almacenar media y la opción “streaming on port 80”. Alternativamente se pueden configurar estos dos parámetros modificando el fichero streamingserver.xml del directorio “/etc/streaming/” añadiendo el puerto 80 a la lista de puertos y poniendo la ruta al directorio “storage” en la propiedad “movie_folder”.

Atención: para evitar problemas con MP4Box, los filepaths han de ser menores que 16 caracteres. Por tanto, se recomienda crear un enlace simbólico a la carpeta “storage” como por ejemplo:

/var/alembik -> /var/lib/services/tomcat/webapps/alembik/storage/

Y utilizarlo a la hora de configurar DSS.

Wurlf:
Este XML ofrece información a *Alembik* sobre cómo transcodificar diferentes tipos de media para los dispositivos en él descritos. Descargar última versión y colocar en cualquier directorio.

http://sourceforge.net/project/showfiles.php?group_id=55408&package_id=50315

Desplegar *Alembik*

Descargar alembik4tomcat de la página oficial de *Alembik*.

Mover los ficheros alembik.war (y alembik-web.war si se quiere probar el servicio en local) a la carpeta webapps del tomcat.

Arrancar el tomcat para desplegar los servicios.

Entrar en el servicio de peticiones (http://localdomain:localport/alembik) y configurar a través de la consola de administración (formulario web).

Introducir la posición de los ejecutables de las aplicaciones externas, la del fichero wurlf, la del directorio de almacenamiento e indicar la versión de *ffmpeg*. Recordar que para las URL será necesario utilizar una ruta a un dominio, y no a localhost.

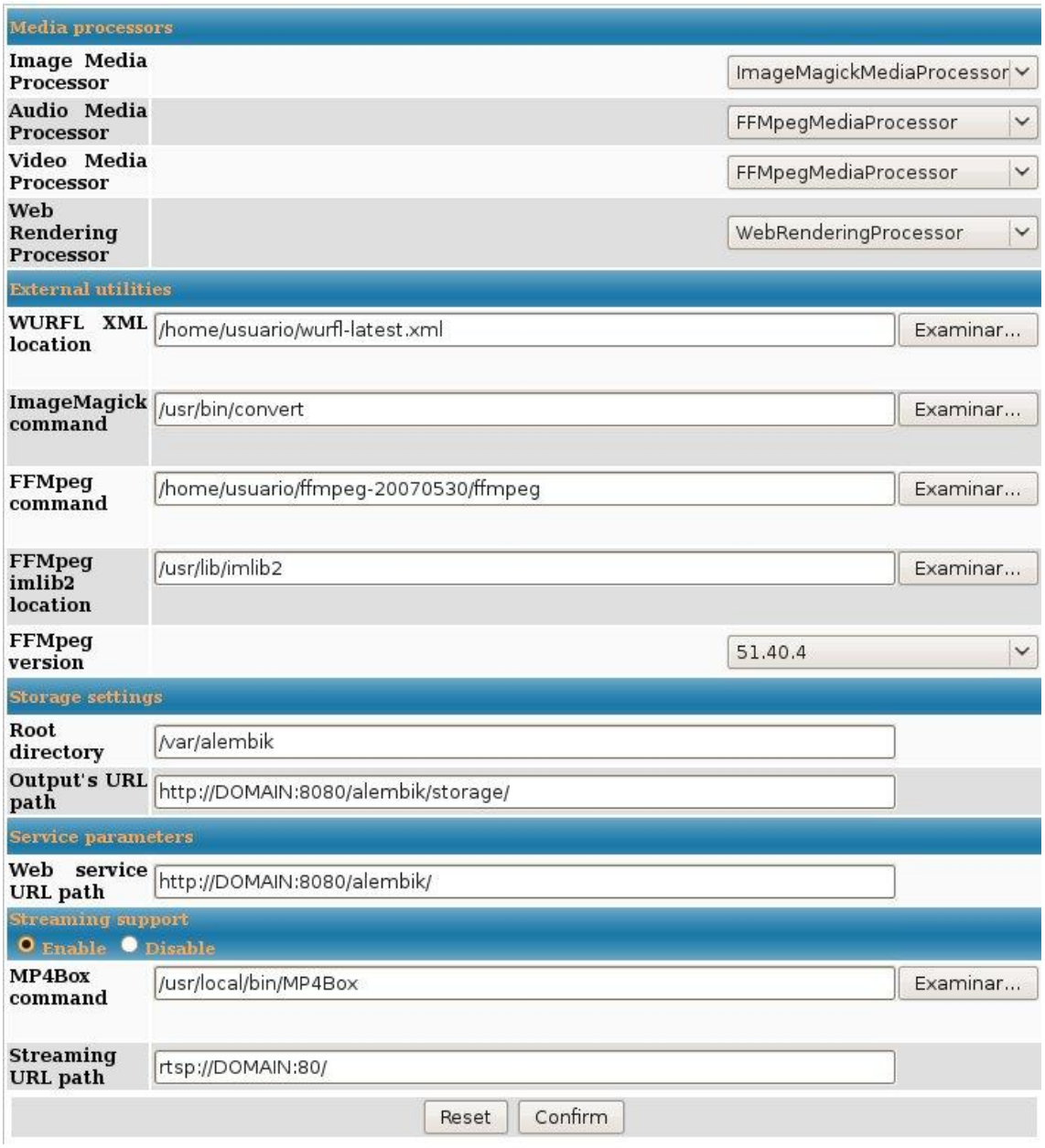

Fig. 1 Consola de administración del *framework Alembik*

Es posible que se produzca un error al no poder resolver el nombre de la máquina por que el fichero /etc/hosts está mal configurado o no existe.

También es posible que la memoria asignada a la máquina virtual de *java* del tomcat sea insuficiente. En ese caso deberá crearse un fichero setenv.sh en la carpeta bin del directorio del tomcat. El contenido de ese fichero será:

JAVA_OPTS="-Xms1024m -Xmx1024m"

Es necesario más de un giga byte de ram para esta configuración. Si la memoria es menor hay que reducir los valores.

Para poder realizar peticiones de tipo text-overlay es necesario crear una variable de entorno FONTPATH que apunte a un directorio con fuentes de tipo TTF y un fichero rgb.txt que contenga las definiciones de los colores en RGB.